Contingent payments on a public ledger: models and reductions for automated verification

Sergiu Bursuc and Steve Kremer

Inria Nancy-Grand'Est & LORIA, France

Abstract. We study protocols that rely on a public ledger infrastructure, concentrating on protocols for zero-knowledge contingent payment, whose security properties combine diverse notions of fairness and privacy. We argue that rigorous models are required for capturing the ledger semantics, the protocol-ledger interaction, the cryptographic primitives and, ultimately, the security properties one would like to achieve.

Our focus is on a particular level of abstraction, where network messages are represented by a term algebra, protocol execution by state transition systems (e.g. multiset rewrite rules) and where the properties of interest can be analyzed with automated verification tools. We propose models for: (1) the rules guiding the ledger execution, taking the coin functionality of public ledgers such as Bitcoin as an example; (2) the security properties expected from ledger-based zero-knowledge contingent payment protocols; (3) two different security protocols that aim at achieving these properties relying on different ledger infrastructures; (4) reductions that allow simpler term algebras for homomorphic cryptographic schemes.

Altogether, these models allow us to derive a first automated verification for ledger-based zero-knowledge contingent payment using the Tamarin prover. Furthermore, our models help in clarifying certain underlying assumptions, security and efficiency tradeoffs that should be taken into account when deploying protocols on the blockchain.

1 Introduction

The blockchain and its associated public ledger promise a practical solution to a basic need for security protocols: a system that operates as stated, providing reliable outcome to all agents. Both deployed [1–4] and abstract [5, 6] ledgers are ordered sequences of states - *state transition systems* respecting operational constraints. The goal of the underlying distributed protocols is to ensure that the ledger is indeed public, unique, alive and consistent. Protocols can then be based on transaction and smart contract semantics - i.e. rules that guide the state transition system - to implement functionality that would otherwise be inefficient or require trusted parties. Take *fair exchange*: two parties want to swap assets according to a contract that ensures fairness : any information or value transfer is reciprocated as planned [7]. The problem can be solved with optimistic assumptions, calling a trusted third party only when needed [8–10], or with digital (counter)cheques and transactions inside multi-party computations [11–13].

A public ledger provides an alternative solution to the problem, specified as a *zero-knowledge contingent payment* (ZKCP) for a seller and buyer. We suppose that the

information of interest can be expressed as data (a *witness*) satisfying functional constraints (a desired *result*), e.g. a sudoku solution respects additive constraints, a prime factor decomposition satisfies multiplicative constraints, etc. ZKCP goals are: for the ***Seller*** - a delivered witness will be paid for; for the ***Buyer*** - a paid for witness will be delivered. Classically, these properties require trust and coordination with third parties. On public ledgers, reliable semantics and dedicated cryptographic protocols can minimize trust and interaction [14–18].

Challenges. Protocol actions occur at distinct levels: from local cryptographic objects, to network transactions, to ledger confirmation. Their respective semantics is useful in protocol design, where parties can agree on desired ledger actions beforehand, yet the concurrent environment opens up new challenges:
• *Multiple sessions, concurrent ledger access.* Asynchronicity leads to ambiguity about what it means to be paid. For example, a seller should ensure it will not be *paid* the same coin for two witnesses. If multiple sessions run in parallel, some with colluding parties, protocol messages may be mixed up and exploited. Valid transaction requests do not necessarily result in confirmed ledger transactions : if the adversary obtains private keys by exploiting the protocol, a race ensues between honest and adversarial messages claiming a coin. Protocols should ensure this does not happen - this is not usually an explicit goal.
• *Transaction finality.* In fact, it is commonly advised to wait for transactions to be finalized on the ledger to ensure payment. Yet, we show that ZKCP protocols (have to) provide a stronger property: as early as a transaction request is being sent over the network, one should ensure that the corresponding coin cannot be spent in any other way, because specific fields from the transaction may help the adversary in revealing secrets - so we cannot afford the transaction to fail.
• *Cryptographic interaction.* Ledger-based protocols produce complex cryptographic objects that engage ledger transitions at the same time as private data transfer, e.g. [15] relies on homomorphic encryption to produce a (secret) ECDSA signature that will perform a ledger transaction; this signature is commited in a zero-knowledge proof ensuring the corresponding ledger transition will furthermore reveal the witness. Such interaction between cryptography and the ledger extends the scope of crypto primitives to new protocols - dedicated, fine-grained security models are needed to evaluate them.
• *Security foundations.* Compounding all of above: ledger-based protocols are network cryptographic protocols executed in an adversarial environment. There is history of attacks and foundations for such protocols - see e.g. [19–23] for recent examples - showing the importance of rigorous security specification and automated verification. Furthermore, we need generic models that allow a clear separation between security properties, ledger infrastructure and cryptographic protocols.

Our contributions address these challenges by formal models connecting the ledger, the ledger-based protocols, the cryptographic primitives and the desired security properties in a specification that can be used as input for automated verification tools. We use the Tamarin prover [24] for verification: it provides an expressive language to specify (cryptographic) state transition systems and to restrict their traces by logical formulas.
• *Public ledger.* We show that the model of the blockchain as a structured computational resource has a natural formal (or symbolic) counterpart combining multiset rewriting,

term algebras and first order logic [24–26]. We identify minimal restrictions on multiset rewriting rules that make them function as a blockchain transition system, i.e. a smart contract. We also show how protocol rules can operate in order to exploit the ledger semantics. We specify the electronic coin functionality provided in e.g. Bitcoin [1] as an example (section 3).

• *ZKCP on public ledgers.* We consider two ZKCP protocols [14, 15] and perform their formal verification in a unified, generic model that captures their different features (sections 4 and 5). The specification tackles a strong attacker that can run multiple sessions, corrupt parties, control the network (in particular drop, reorder, replace the messages to the ledger) and exploit the cryptographic properties of messages. The formal security properties clearly circumscribe the expected ZKCP guarantees, both in their positive and in their negative aspects: e.g. a buyer will learn the witness or otherwise it can obtain a refund; a seller will obtain payment, unless there is a delivery delay to the ledger; etc. The security properties are parametric, so that different protocols can accordingly instantiate the notions of payment, time delay, witness extraction, etc.

• *Advanced cryptography.* The protocol we consider in section 5 aims at a basic version of Bitcoin, with a minimal scripting language for signature verification; this calls for complex cryptography, intertwining homomorphic encryption, randomized signatures, diffie-hellman exponentiation and specialized zero-knowledge proofs. The corresponding formal specification as a message theory is out of the scope for any current automated verification tools. We provide a theoretical framework and a reduction result showing that it is sound to consider a simplified theory as input (section 6). We start from a general theory where some of the function symbols are homomorphic: from $f(u, \overline{w})$ and v, one can derive $f(u*v, \overline{w})$, where $*$ is the product in an abelian group. In the reduced theory: 1) the homomorphic properties are restricted as follows: the adversary can derive $f(u*v, \overline{w})$ from $f(u, \overline{w})$ only if u is a product of messages created by honest parties; 2) the abelian group is degenerated: the adversary can derive the factors $u_1, \ldots, u_k$ of any product $u_1 * \ldots * u_k$, without being required to know any inverse.

2 Preliminaries: computation model

Term algebra [27]. $\mathcal{F}$ denotes the set of function symbols and $\mathcal{F}^{(n)}$ those of arity n. The set of terms built from $\mathcal{F}$, a set of names and a set of variables is $\mathcal{T}$. Tuples of terms are denoted by an overline, e.g. $\overline{u} = (u_1, \ldots, u_n)$. We let $st(t)$ be the subterms of a term t, and $top(t)$ be its top symbol. $\mathcal{F}$ is endowed with a rewrite system: a set of rewrite rules $\mathcal{R}$, that we denote by $l \to r$, modulo a set of equations $\mathcal{E}$, that we denote by $l \approx r$. $\mathcal{R}$ or $\mathcal{E}$ can be empty. For a term t, $t{\downarrow_{\mathcal{R}}}$ is its normal form, obtained after applying all possible rewrite steps (modulo $\mathcal{E}$) from $\mathcal{R}$. Implicitly, terms are normalized and term equalities interpreted modulo $(\mathcal{R}, \mathcal{E})$.

Example 1. For the theory of randomized signatures, as instantiated e.g. by (EC)DSA [28], we let $\mathcal{F}_{\mathsf{sig}} = \{\mathsf{sign}, \mathsf{ver}, \mathsf{ok}, g\}$ and $\mathcal{R}_{\mathsf{sig}}$ be the signature verification rule:$\mathsf{ver}(\mathsf{sign}(x, y, z), x, g(y)) \to \mathsf{ok}$. Here $g(y)$ represents the public key corresponding to a secret key y, i.e. the group element that corresponds to raising a group generator g to a scalar power y. The third argument of sign takes the role of the randomness: $\mathsf{sign}(m, k, r_1)$ and $\mathsf{sign}(m, k, r_2)$ are two distinct signatures of m with key k.

The theory of an abelian group (AG), e.g. $\mathbb{Z}_q$, is modeled by the signature $\mathcal{F}_* = \{*, i\}$ and the set of equations $\mathsf{AG} = \{x * i(x) \approx 1, x * 1 \approx x\} \cup \mathsf{AC}$ where $\mathsf{AC} = \{x * y \approx y * x, (x * y) * z \approx x * (y * z)\}$ models associativity and commutativity.

Multiset rewriting and state transitions [24, 26]. The signature is extended with fact symbols to represent 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 by the protocol, e.g. for representing the state. These symbols can be persistent (the corresponding facts cannot disappear), or linear (the corresponding facts are consumed by rules and protocol rules can update them). Persistent fact symbols are prefixed by !, e.g. $!\mathbf{F}$. A multiset can contain multiple copies of the same linear fact.

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

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.

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

The semantics ensures that y and z above are instantiated to public, resp. fresh names.

Intruder deduction rules are of the form $[\mathsf{K}(u_1), \ldots, \mathsf{K}(u_k)] \Rightarrow [\mathsf{K}(v)]$ - defining operations on messages. These are typically $[\mathsf{K}(x_1), \ldots, \mathsf{K}(x_k)] \Rightarrow [\mathsf{K}(f(x_1, \ldots, x_k))]$ for all $f \in \mathcal{F}^{(k)}$. We also allow more general deduction rules, as in Example 2 and Figure 4. Such rules can wlog replace rewrite rules $f(l_1, \ldots, l_k) \rightarrow r$ for symbols f with no other occurence in the rewrite system and whose occurence in protocol rules is not under a term context. An *intruder theory*, that we denote by $\mathcal{I}$, is thus given by a set of intruder deduction rules plus $(\mathcal{R}, \mathcal{E})$. For a set of terms $\{t_1, \ldots, t_n, t\}$ we let $\{t_1, \ldots, t_n\} \vdash t$ if $\mathsf{K}(t)$ can be obtained from $\mathsf{K}(t_1), \ldots, \mathsf{K}(t_n)$ using intruder deduction rules. *Protocol rules* model the execution of the protocol by honest parties. There are basic restrictions ensuring that protocol rules are a sound model of protocol executions [26]; we will follow them implicitly in our models and examples.

Example 2. Exponentiation in a Diffie-Hellman group can be represented by the rewrite rule $exp(g(x), y) \rightarrow g(x * y)$ together with the deduction rule $[\mathsf{K}(x_1), \mathsf{K}(x_2)] \Rightarrow$

$[\mathsf{K}(exp(x_1, x_2))]$. Alternatively, the deduction rule $[\mathsf{K}(g(x)), \mathsf{K}(y)] \Rightarrow [\mathsf{K}(g(x * y))]$ allows to model the corresponding operation performed by the attacker (without requiring explicit application of exp). Similarly, a protocol rule can directly perform exponentiation without explicit use of the symbol exp, e.g. $[\mathsf{In}(g(x)), \mathsf{Fr}(y)] \Rightarrow [\mathsf{Out}(g(x * y))]$.

For a rule P, we let $facts(P)$, $in(P)$, $out(P)$, $lhs(P)$, $rhs(P)$, $act(P)$ be respectively the set of all facts, of input facts (e.g. $\mathsf{In}(u)$), of output facts (e.g. $\mathsf{Out}(u)$), of left-hand side facts (i.e. premisses), of right-hand side facts (i.e. conclusions) and of action facts. For a set of facts $\mathbf{F}$, we let $msg(\mathbf{F})$ be the set of messages that are arguments of facts in $\mathbf{F}$. We let $io(P) = msg(in(P) \cup out(P))$.

Traces and properties. A *trace* τ is a sequence of applications of $n \geq 1$ msr rules, interleaving applications of protocol, intruder and network deduction rules. For every $i \in \{1, \ldots, n\}$, we let P_i be the rule applied at step i and θ_i be the corresponding substitution. We define:

- $facts(\tau, i) = act(P_i)\theta_i{\downarrow}$ if P_i is a protocol or network deduction rule;
- $facts(\tau, i) = \{\mathsf{K}(v\theta_i{\downarrow})\}$ if P_i is an intruder deduction rule with $rhs(P_i) = \{\mathsf{K}(v)\}$

For a set of rules Q, we denote by $traces(Q)$ the set of all valid traces that can be derived from elements in Q. 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 term equality $t_1 \approx t_2$, or a timepoint ordering $i < j$, or a timepoint equality $i = j$, or an action fact $\mathbf{F}@i$ for a fact $\mathbf{F}$ and timepoint i. A *trace formula* is a first-order logic formula obtained from trace atoms by applying the usual quantification and logical connectives. Given a trace τ and trace formula ϕ, whose variables are all bound, the satisfaction relation $\tau \models \phi$, is defined recursively as expected, in particular $\tau \models \mathbf{F} \mathbin{@} i$ iff $\mathbf{F} \in facts(\tau, i)$.

For a set of rules Q and trace formulas Ψ, Φ, we let $Q \models \Phi$ iff $\forall \tau \in traces(Q).\ \tau \models \Phi$ and $Q; \Psi \models \Phi$ iff $\forall \tau \in traces(Q).\ \tau \models \Psi \Rightarrow \Phi$. For verification, $(Q; \Psi)$ will be a system specification and Φ a property to verify; Q defines local transition rules, while Ψ defines additional, global restrictions on the set of traces for the specified system.

Example 3. Consider the binary fact symbol Eq and the formula

$$\Psi_{\mathsf{eq}} : \forall x, y, i.\ \mathsf{Eq}(x, y) \mathbin{@} i \Rightarrow x \approx y.$$

An $\mathsf{Eq}(u, v)$ action in a rule allows then to test that $u \approx_{\mathcal{E}} v$ before proceeding. Take $P = [\mathsf{In}(u), \mathsf{In}(v), \mathsf{Fr}(s)] \mathrel{-}[\ \mathsf{Eq}(u, v)\ ]\mathrel{\rightarrow} [\mathsf{Out}(s)]$. Then $\mathsf{K}(a), \mathsf{K}(a), \mathsf{Eq}(a, a), \mathsf{K}(s)$ is a trace of P satisfying Ψ_{eq}, while $\mathsf{K}(a), \mathsf{K}(f(a)), \mathsf{Eq}(a, f(a)), \mathsf{K}(s)$ does not.

Consider the unary symbol Fresh and the restriction

$$\Psi_{\mathsf{fresh}} : \forall x, i, j.\ \mathsf{Fresh}(x) \mathbin{@} i \ \wedge\ \mathsf{Fresh}(x) \mathbin{@} j \Rightarrow i = j.$$

It ensures that every occurrence of $\mathsf{Fresh}(t)$ is with a different t. Assume we add $\mathsf{Fresh}(\langle u, v \rangle)$ as an action in P. Then, among $traces(P)$, $\ldots \mathsf{Eq}(a, a), \ldots, \mathsf{Eq}(a, a)$ does not satisfy Ψ_{fresh}, while $\ldots \mathsf{Eq}(a, a), \ldots, \mathsf{Eq}(b, b)$ does.

Example 4. Consider the set of rules Q_{keys}:

- $[\mathsf{Fr}(k)]\!-\!\![!\mathsf{Key}(k)\ ]\!\!\rightarrow[!\mathsf{Pk}(g(k)), !\mathsf{Key}(k), \mathsf{Out}(g(k))]$
- $[!\mathsf{Key}(x)]\!-\!\![\ \mathsf{Corrupt}(g(x))\ ]\!\!\rightarrow[\mathsf{Out}(x)]$

It models a basic key infrastructure. The formula $\Phi : !\mathsf{Key}(x) \ @\ i \Rightarrow \neg\exists j.\mathsf{K}(x) \ @\ j$ says that keys are secret. Then $Q_{\mathsf{keys}} \not\models \forall x, i.\Phi$, since the second rule in Q_{keys} allows the attacker to corrupt keys. Now consider the protocol rule

$$Q_{\mathsf{sign}} : [\mathsf{Fr}(a), !\mathsf{Key}(x)]\!-\!\![\ \mathsf{Honest}(g(x)), \mathsf{Sign}(x)\ ]\!\!\rightarrow[\mathsf{Out}(\mathsf{sign}(a, k, \rho_r))]$$

the formula $\Phi' :\ \mathsf{Sign}(x) \ @\ j \Rightarrow \neg\exists j.\mathsf{K}(x) \ @\ j$ - saying that keys used in Q_{sign} are secret - and the restriction: $\Psi_{\mathsf{hon}} : \forall x, i.\ \mathsf{Honest}(x) \ @\ i \Rightarrow \neg\exists j.\ \mathsf{Corrupt}(x) \ @\ j$. Then we have $Q_{\mathsf{keys}}, Q_{\mathsf{sign}}; \Psi_{\mathsf{hon}} \models \forall x, i.\Phi'$ because we have added the restrictions that keys in Q_{sign} are honest and that honest keys cannot be corrupted.

Public data. Tamarin allows the use of variables that can be instantiated only with messages of a public sort. They are denoted by $\$x$, and can occur anywhere in a protocol msr rule. As in Example 4, we will use annotations of ρ for such data, e.g. ρ_r for a public nonce, ρ_{sn} for a serial number, etc.

Protocol state. Specifications rely on sequences of protocols rules $(P_0, \ldots, P_k)$, where each rule P_i should be executed before P_{i+1} and can pass on, via facts, state data to P_{i+1}. To avoid clutter, we use a symbol state_i to represent this transmission, and we allow P_{i+1} to reference any variables from P_i that should be formally passed via state facts. We denote by $\mathsf{state}_i\lceil x = u\rfloor$ the pattern matching of state variable x by a term u.

3 Public ledgers: facts, rules, coins

Coin ledger. The protocols we consider are based on coin contracts of e.g. Bitcoin [1]: a *coin* is represented by an object $(\mathsf{sn}, g(k))$ on the ledger, where sn is a serial number, and $g(k)$ is the public key of the coin owner. Serial numbers are computed as the hash of the transaction that created the coin; for simplicity, we assume they are fresh public numbers. To spend a coin, i.e. transfer it to a new owner, the ledger expects a transaction request, attested by a signature from the current owner, containing the sn of the coin to be spent, the public key $g(k')$ of the new owner and (implicitly) the serial number sn' of the new coin. If the signature is valid, the coin $(\mathsf{sn}, g(k))$ is marked as spent, and a new coin $(\mathsf{sn}', g(k'))$ is created for the new owner. We call *basecoins* these coins.

We will also make use of *hashcoins*: *hashed timelock contracts* [29] used to establish trust relationships outside the ledger [30, 31]. They perform a transaction by which one of the two parties, say A, obtains the preimage of a hash - which can e.g. be a key encrypting some data of interest - while the other party, say B, provides the hash preimage and obtains a basecoin in return. A performs a ledger transaction pledging one of A's coins into a hashcoin, providing the desired hash image and the public key of B. B can then claim the coin using a (signed) inverse of the image. A timeout mechanism ensures the coin can be returned to A if there was no action from B in due time. A hashcoin can be represented by a tuple $(\mathsf{sn}, g(k), h(x), g(k'))$ here $g(k)$ represents the coin creator, who can obtain it after timeout, $h(x)$ is the desired hash image, and $g(k')$ is the party that can claim sn by supplying x.

Formal model. We consider two special sets of disjoint fact symbols: one for *ledger facts*, denoted by $\mathbf{F}_{\mathcal{L}}$, and one for *check facts*, denoted by $\mathbf{F}_{\mathcal{C}}$. Ledger facts will be used to represent the state of the ledger. For example, they can record who is the owner of an asset, what are the elements of a given transaction, etc. Ledger facts are assumed persistent because the ledger history cannot change. Check facts, on the other hand, will be used by protocols to restrict their executions with respect to the (current or past) states of the ledger. For example, they can be used to ensure that a coin, whose existence is recorded by a ledger fact, has not yet been spent.

Example 5. Let $\mathcal{F}_{\mathcal{L}}^{\mathsf{coin}} = \{!\mathsf{Coin}, !\mathsf{HCoin}, !\mathsf{Spend}, !\mathsf{Time}\}$ and $\mathcal{F}_{\mathcal{C}}^{\mathsf{coin}} = \{\mathsf{Unspent}\}$. The corresponding facts represent: $!\mathsf{Coin}(\mathsf{sn}, g(k))\ @\ i$ - a coin sn created at timepoint i belonging to the public key $g(k)$; $!\mathsf{HCoin}(\mathsf{sn}, \langle g(k_1), g(k_2), h(t)\rangle)\ @\ i$ - a hashcoin sn that can be claimed for $g(k_2)$ by supplying t and a signature, or for $g(k_1)$ after timeout by supplying a signature; $!\mathsf{Spend}(\mathsf{sn}, u, w, v)\ @\ i$ - the transfer of a coin (sn, u) to a new owner v at timepoint i, relying on supporting data w: w is a signature when sn is a basecoin, plus possibly a hash preimage when sn is a hashcoin; $!\mathsf{Time}(\mathsf{sn})\ @\ i$ marks the fact that the hashcoin sn was reclaimed after a timeout at timepoint i; $\mathsf{Unspent}(\mathsf{sn})\ @\ i$ checks the ledger to ensure the coin sn is unspent at i.

The semantics of the ledger is defined by msr rules that can only be triggered by ledger facts and public inputs, and can only produce ledger facts and public outputs. Ledger restrictions ensure additional constraints for the states produced by the ledger. These rules and constraints define the ledger state transition system and make it available for external protocols, which may be executed by honest or adversarial parties.

Definition 1. *A msr rule P is a* ledger rule *if: (1) $facts(P) \subseteq in(P) \cup out(P) \cup \mathbf{F}_{\mathcal{L}}$; (2) $rhs(P) \subseteq act(P)$. P is* ledger-respecting *if $(act(P) \cup rhs(P)) \cap \mathbf{F}_{\mathcal{L}} = \emptyset$. A* ledger restriction *is a trace formula with facts in $\mathbf{F}_{\mathcal{L}} \cup \mathbf{F}_{\mathcal{C}}$.*

Properties of ledger rules in Definition 1 ensure that: (1) the ledger transition system depends only on ledger facts and public inputs; (2) all produced ledger facts are recorded as actions in the trace. In this paper we consider public ledgers, e.g. [1–4], so the ledger rules will also satisfy (3) $msg(rhs(P)) \subseteq msg(out(P))$. This is not an inherent restriction of the model, and partially public ledgers, e.g. [32], may be considered in the scope of Definition 1. Bearing in mind the properties (2) and (3) of our considered ledger rules, in order to simplify the presentation of our examples in the paper, we will avoid duplication, writing $[F_0]\!-\![\ \Phi\ ]\!\rightarrow\![F_1]$ instead of $[F_0]\!-\![\ \Phi, F_1\ ]\!\rightarrow\![F_1, \mathsf{Out}(msg(F_1))]$ as expected. All protocol rules will be ledger-respecting as in Definition 1, so the only way to produce ledger facts is by passing through ledger rules; on the other hand, protocol rules can freely access ledger facts to check the state of the ledger, so we can have $lhs(P) \cap \mathbf{F}_{\mathcal{L}} \neq \emptyset$.

In Fig. 1, the rule $\mathsf{R}_{\mathsf{new}}$ abstracts the coin mining process; the other rules model formally the coin transactions as described above: spending coins to coins, to hashcoins, and back to coins. The rule $\mathsf{R}_{\mathsf{h2cr}}$ produces a ledger fact $!\mathsf{Time}(x_{\mathsf{sn}})$ to record that the corresponding coin was reclaimed after a timeout. The rules $\mathsf{S}_{\mathsf{c2h}}, \mathsf{S}_{\mathsf{h2c}}$ assume Hash and Inv to be defined by their context as a hash image of interest and a hash preimage.

Fig. 1. ***Ledger coin rules:*** $\mathcal{L}_{\mathsf{base}} = \{\mathsf{R}_{\mathsf{new}}, \mathsf{R}_{\mathsf{c2c}}\};\ \mathcal{L}_{\mathsf{hash}} = \mathcal{L}_{\mathsf{base}} \uplus \{\mathsf{R}_{\mathsf{c2h}}, \mathsf{R}_{\mathsf{h2c}}, \mathsf{R}_{\mathsf{h2cr}}\}$

$\mathsf{R}_{\mathsf{new}} : [!\mathsf{Pk}(x_{\mathsf{pk}}), \mathsf{In}(\langle s, x_{\mathsf{sn}}\rangle)] \multimap\!\{\ \mathsf{ver}(s, x_{\mathsf{sn}}, x_{\mathsf{pk}}) \approx \mathsf{ok}\ \}\!\mapsto [!\mathsf{Coin}(x_{\mathsf{sn}}, x_{\mathsf{pk}})]$

$\mathsf{R}_{\mathsf{c2c}} : [!\mathsf{Coin}(x_{\mathsf{sn}}, x_{\mathsf{pk}}), \mathsf{In}(u)] \multimap\!\{\ \Phi_{\mathsf{c2c}}(x_{\mathsf{sn}}, x_{\mathsf{pk}}, u)\ \}\!\mapsto [!\mathsf{Spend}(x_{\mathsf{sn}}, x_{\mathsf{pk}}, v), !\mathsf{Coin}(y_{\mathsf{sn}}, y_{\mathsf{pk}})]$

$\mathsf{R}_{\mathsf{c2h}} : [!\mathsf{Coin}(x_{\mathsf{sn}}, x_{\mathsf{pk}}), \mathsf{In}(u)] \multimap\!\{\ \Phi_{\mathsf{c2h}}(x_{\mathsf{sn}}, x_{\mathsf{pk}}, u)\ \}\!\mapsto [!\mathsf{Spend}(x_{\mathsf{sn}}, x_{\mathsf{pk}}, s, y), !\mathsf{HCoin}(y_{\mathsf{sn}}, y)]$

$\mathsf{R}_{\mathsf{h2c}} : [!\mathsf{HCoin}(x_{\mathsf{sn}}, y), \mathsf{In}(u)] \multimap\!\{\ \Phi_{\mathsf{h2c}}(x_{\mathsf{sn}}, y, u)\ \}\!\mapsto [!\mathsf{Spend}(x_{\mathsf{sn}}, y, s, y_{\mathsf{pk}}), !\mathsf{Coin}(z_{\mathsf{sn}}, y_{\mathsf{pk}})]$

$\mathsf{R}_{\mathsf{h2cr}} : [!\mathsf{HCoin}(x_{\mathsf{sn}}, y), \mathsf{In}(u)] \multimap\!\{\ \Phi_{\mathsf{h2cr}}(x_{\mathsf{sn}}, y, u)\ \}\!\mapsto [\ldots, !\mathsf{Coin}(z_{\mathsf{sn}}, x_{\mathsf{pk}}), !\mathsf{Time}(x_{\mathsf{sn}})\ ]$

where

$\mathsf{R}_{\mathsf{c2c}} : u = \langle s, y_{\mathsf{sn}}, y_{\mathsf{pk}}\rangle; \Phi_{\mathsf{c2c}} = \mathsf{ver}(s, \langle \mathsf{c2c}, x_{\mathsf{sn}}, y_{\mathsf{sn}}, y_{\mathsf{pk}}\rangle, x_{\mathsf{pk}}) \approx \mathsf{ok}; v = \langle s, y_{\mathsf{pk}}\rangle$

$\mathsf{R}_{\mathsf{c2h}} : u = \langle s, y_{\mathsf{sn}}, y_{\mathsf{pk}}, y_h\rangle; \Phi_{\mathsf{c2h}} = \mathsf{ver}(s, \langle \mathsf{c2h}, x_{\mathsf{sn}}, y_{\mathsf{pk}}, y_h\rangle, x_{\mathsf{pk}}) \approx \mathsf{ok}; y = \langle x_{\mathsf{pk}}, y_{\mathsf{pk}}, y_h\rangle$

$\mathsf{R}_{\mathsf{h2c}} : y = \langle x_{\mathsf{pk}}, y_{\mathsf{pk}}, y_h\rangle; u = \langle s, y_{\mathsf{sn}}, y_w\rangle;$
$\Phi_{\mathsf{h2c}} = \mathsf{ver}(s, \langle \mathsf{h2c}, x_{\mathsf{sn}}, y_w\rangle, y_{\mathsf{pk}}) \approx \mathsf{ok} \wedge y_h \approx h(y_w)$ ***(similarly for*** $\mathsf{R}_{\mathsf{h2cr}}$***)***

Ledger-based protocol rules (typical examples)

$\mathsf{S}_{\mathsf{c2c}} : [\ !\mathsf{Key}(x_{sk}), !\mathsf{Pk}(y_{\mathsf{pk}}), !\mathsf{Coin}(x_{\mathsf{sn}}, g(x_{sk})), x_s = \mathsf{sign}(\langle \mathsf{c2c}, x_{\mathsf{sn}}, \rho_{\mathsf{sn}}, y_{\mathsf{pk}}\rangle, x_{sk}, \rho_r)\ ]$
$\multimap\!\{\ \mathsf{Unspent}(x_{\mathsf{sn}})\ \}\!\mapsto [\ \mathsf{Out}(\langle x_s, \rho_{\mathsf{sn}}, y_{\mathsf{pk}}\rangle)\ ]$

$\mathsf{S}_{\mathsf{c2h}} : [\ !\mathsf{Key}(x_{sk}), !\mathsf{Pk}(y_{\mathsf{pk}}), !\mathsf{Coin}(x_{\mathsf{sn}}, g(x_{sk})), \mathsf{Hash}(y_h)\ ]$
$\multimap\!\{\ \mathsf{Unspent}(x_{\mathsf{sn}})\ \}\!\mapsto [\ \mathsf{Out}(u_{\mathsf{c2h}})\ ]$

$\mathsf{S}_{\mathsf{h2c}} : [\ !\mathsf{Key}(y_{sk}), !\mathsf{HCoin}(x_{\mathsf{sn}}, \langle x_{\mathsf{pk}}, g(y_{sk}), h(x_w)\rangle), \mathsf{Inv}(y_w)\ ]$
$\multimap\!\{\ \mathsf{Unspent}(x_{\mathsf{sn}}), \mathsf{Claim}(x_{\mathsf{sn}}, g(y_{sk}))\ \}\!\mapsto [\ \mathsf{Out}(u_{\mathsf{h2c}})\ ]$

where $t_{\mathsf{c2h}} = \langle \mathsf{c2h}, x_{\mathsf{sn}}, y_{\mathsf{pk}}, y_h\rangle\ ;\ u_{\mathsf{c2h}} = \langle \mathsf{sign}(t_{\mathsf{c2h}}, x_{sk}, \rho_r), \rho_{\mathsf{sn}}, y_{\mathsf{pk}}, y_h\rangle$
$t_{\mathsf{h2c}} = \langle \mathsf{h2c}, x_{\mathsf{sn}}, x_w, \rho_{\mathsf{sn}}\rangle\ ;\ u_{\mathsf{h2c}} = \langle \mathsf{sign}(t_{\mathsf{h2c}}, y_{sk}, \rho_r), \rho_{\mathsf{sn}}, y_w\rangle$

Ledger restrictions define additional constraints that should be satisfied by the public ledger. If $facts(\Phi) \subseteq \mathbf{F}_{\mathcal{L}}$ then the restriction Φ is *inherent to the semantics of the ledger*, i.e. it is a check performed by the (distributed) trusted party that builds the ledger. On the other hand, if $\exists \mathbf{F} \in facts(\Phi) \cap \mathbf{F}_{\mathcal{C}}$, then Φ *restricts the execution of the protocols* with respect to the public ledger: a protocol rule P with a substitution θ such that $\mathbf{F}\theta \in act(P\theta)$ can perform a transition at timepoint i, only if $\mathbf{F}\theta\ @\ i$ is consistent with $\Phi\theta$ and the previous ledger facts.

Example 6. The following formulas define ledger restrictions for coins on $\mathcal{L}_{\mathsf{base}}, \mathcal{L}_{\mathsf{hash}}$

$$\Psi_0 : \forall x, \overline{y}, \overline{z}, i, j.\ !\mathsf{Spend}(x, \overline{y})\ @\ i \wedge !\mathsf{Spend}(x, \overline{z})\ @\ j \Rightarrow i = j\ \wedge \overline{y} = \overline{z}$$
$$\Psi_1 : \forall x, y, z, i, j.\ !F_1(x, y)\ @\ i \wedge !F_2(x, z)\ @\ j\ \Rightarrow i = j\ \wedge\ y = z \quad (\forall F_1, F_2 \in \{\mathsf{Coin}, \mathsf{HCoin}\})$$
$$\Psi_2 : \forall x, \overline{y}, i, j.\ \mathsf{Unspent}(x)\ @\ i\ \wedge\ !\mathsf{Spend}(x, \overline{y})\ @\ j \Rightarrow i < j$$

They ensure that - no coin can be spent twice (Ψ_0); - every fresh coin has a fresh serial number (Ψ_1); - Unspent can hold at timepoint i only if the corresponding coin has not already been spent on the ledger (Ψ_2). Note that Ψ_0, Ψ_1 are inherent ledger restrictions, while Ψ_2 is a protocol ledger restriction. We let $\Psi_{\mathsf{coin}} = \Psi_0 \wedge \Psi_1 \wedge \Psi_2$.

4 Zero knowledge contingent payments

We specify in a general framework the security guarantees that parties can expect from ZKCP protocols. We allow several parameters in definitions, that can be instantiated

differently by specific protocols and ledgers - we illustrate it on $\mathcal{L}_{\mathsf{base}}$ and $\mathcal{L}_{\mathsf{hash}}$. We are interested in generic ZKCP protocols, where any functionality can be obtained by instantiating the protocol with a specific function f. Security is independent of the actual function f, so we consider a generic f in the following.

For intuition, consider first a protocol on $\mathcal{L}_{\mathsf{hash}}$ [14,16]. It assumes a zero-knowledge proof system showing that a ciphertext provided by a party contains a witness for a desired result, where the symmetric encryption key is the preimage of a given hash value. We represent such a proof by $zk(w, v, u)$ where w is the witness, v is the hash preimage used as symmetric key, and u is the secret key of the party constructing the proof (for brevity, we ommit public data that may be part of the proof). The following rewrite rules represent symmetric encryption and zk proof verification:
$\mathsf{sdec}(\mathsf{senc}(x, y), y) \to x \qquad \mathsf{ver}_{\mathsf{zk}}(\mathsf{zk}(x, y, z), \mathsf{senc}(x, y), f(x), h(y), g(z)) \to \mathsf{ok}.$
These define $\mathcal{I}_{\mathsf{hash}}$, where also $\forall f \in \mathcal{F}^{(k)}.\ [\mathsf{K}(x_1), \ldots, \mathsf{K}(x_k)] \Rightarrow [\mathsf{K}(f(x_1, \ldots, x_k))]$.
Assume a seller with private key ks wants to sell w to a buyer with public key $g(kb)$.
Seller 1: generate a fresh key k; output $\mathsf{senc}(w, k), h(k), g(ks), zk(w, k, ks)$;
Buyer 1: receive above data from seller and, if the zk proof verifies, invoke $\mathsf{R}_{\mathsf{c2h}}$ on $\mathcal{L}_{\mathsf{hash}}$ to create a hashcoin for the given $h(k)$ and $g(ks)$: $!\mathsf{HCoin}(\mathsf{sn}, \langle g(kb), g(ks), h(k)\rangle)$;
Seller 2: inspect $\mathcal{L}_{\mathsf{hash}}$ to see if the above coin was created; invoke $\mathsf{R}_{\mathsf{h2c}}$ with k and ks to claim the coin; this reveals k and thus reveals the witness;
Buyer 2: inspect $\mathcal{L}_{\mathsf{hash}}$ to see if $\mathsf{R}_{\mathsf{h2c}}$ was invoked for the created hashcoin; if yes, the ledger will also contain the key k that allows the decryption of the ciphertext received at step 1; if not, the rule $\mathsf{R}_{\mathsf{h2cr}}$ can be invoked after a time delay so that the coin is returned to the original owner.
Timeout. The fairness properties for the ZKCP protocols will be relative to the timely execution of certain operations. More precisely, if a certain action is not performed by a party in due time, then there is another action - grounded on the semantics of the ledger as in Example 7 or on cryptographic primitives as in Example 8 - that can be performed in order to compensate for the missing action.

Example 7 (Ledger timeout). Consider the rule $\mathsf{R}_{\mathsf{h2cr}}$ from Figure 1 modeling the refund of a hashcoin after a timeout. The execution of this rule at timepoint i is accompanied on the ledger by the fact $!\mathsf{Time}(x_{\mathsf{sn}})\ @\ i$ to record that this coin was spent due to a timeout. This allows to specify the possible effects of invoking $\mathsf{R}_{\mathsf{h2c}}$ on $\mathcal{L}_{\mathsf{hash}}$: either the transaction completes as expected, or there was a timeout, i.e. $\mathsf{R}_{\mathsf{h2cr}}$ was invoked. Consider the rule $\mathsf{S}_{\mathsf{h2c}}$ from Figure 1; note the Claim action. Then $\mathcal{L}_{\mathsf{hash}}$ ensures the following property:

$$\forall x, y, z, z_1, z_2, i, j.\ \begin{array}{c}\mathsf{Claim}(x, y)\ @\ i \wedge \\ !\mathsf{Spend}(x, z_1, z_2, z)\ @\ j\end{array} \Rightarrow \begin{array}{c} z = y \vee \\ !\mathsf{Time}(x)\ @\ j\end{array}$$

where $z = y$ happens in a normal execution, and $!\mathsf{Time}(x)\ @\ j$ if the timeout occurs.

Example 8 (Cryptographic timeout [33, 34]). Time commitment schemes allow to produce a commitment to a message that keeps it secret for a period of time. We represent a time commitment to u by $tcom(u)$ and consider the following rule $\mathsf{Q}_{\mathsf{tcom}}$: $[\mathsf{In}(tcom(x))] \multimap [!\mathsf{Time}(x)] \mapsto [\mathsf{Out}(x)]$. We express that fresh committed data is either secret, or it was released after a timeout. Let $P : [\mathsf{Fr}(s)] \multimap [\ \mathsf{Tcom}(s)\ ] \mapsto [\mathsf{Out}(tcom(s))]$. Then $\mathsf{Q}_{\mathsf{tcom}}, P \models \forall x, i, j.\ \mathsf{Tcom}(x)\ @\ i\ \wedge \mathsf{K}(x)\ @\ j\ \Rightarrow\ \exists k.\ k < j \wedge !\mathsf{Time}(x)\ @\ k$

Fig. 2. Formal ZKCP on $\mathcal{L}_{\mathsf{hash}}$; $\mathsf{Seller} = (S_0, S_1, S_2)$; $\mathsf{Buyer} = (B_0, B_1, B_2^{\mathsf{go}}, B_2^{\mathsf{ab}})$

S_0**:** $[\ !\mathsf{Key}(x_{\mathsf{ks}}),\ !\mathsf{Witn}(x_{\mathsf{wtn}})\ ] \multimap [\ \mathsf{Sell}(g(x_{\mathsf{ks}}), x_{\mathsf{wtn}})\ ] \mapsto [\ \mathsf{state}_0\ ]$

S_1**:** $[\ \mathsf{state}_0, \mathsf{Fr}(k), x_{\mathsf{ew}} = senc(x_{\mathsf{wtn}}, k), x_\pi = \mathsf{zk}(x_{\mathsf{wtn}}, k, x_{\mathsf{ks}})\ ] \Rightarrow [\ \mathsf{Out}(\langle x_\pi, x_{\mathsf{ew}}, h(k)\rangle), \mathsf{state}_1\ ]$

S_2**:** $[\ \mathsf{state}_1, !\mathsf{HCoin}(x_{\mathsf{sn}}, \langle x_{\mathsf{pkb}}, g(x_{\mathsf{ks}}), h(k)\rangle)\ ] \multimap [\ \mathsf{Unspent}(x_{\mathsf{sn}}), \mathsf{Claim}(g(x_{\mathsf{ks}}), x_{\mathsf{wtn}}, x_{\mathsf{sn}}, x_{\mathsf{sn}})\ ] \mapsto$
$[\ \mathsf{Out}(\langle \mathsf{sign}(\langle \mathsf{h2c}, x_{\mathsf{sn}}, \rho_{\mathsf{sn}}, k\rangle, x_{\mathsf{ks}}), k, \rho_{\mathsf{sn}}\rangle)\ ]$

B_0**:** $[\ !\mathsf{Res}(x_{\mathsf{res}}), !\mathsf{Key}(x_{\mathsf{kb}}), !\mathsf{Pk}(x_{\mathsf{pks}}), !\mathsf{Coin}(x_{\mathsf{sn}}, g(x_{\mathsf{kb}}))\ ] \Rightarrow [\ \mathsf{state}_0\ ]$

B_1**:** $[\ \mathsf{state}_0, \mathsf{In}(\langle x_\pi, x_{\mathsf{ew}}, x_h\rangle)\ ] \multimap [\ \mathsf{ver}_{\mathsf{zk}}(x_\pi, x_{\mathsf{ew}}, x_{\mathsf{res}}, x_h, x_{\mathsf{pks}}) \approx \mathsf{ok},$
$\mathsf{Pay}(g(x_{\mathsf{kb}}), x_{\mathsf{res}}, \rho_{\mathsf{sn}}, \langle x_\pi, x_{\mathsf{ew}}, x_h\rangle)\ ] \mapsto [\ \mathsf{Out}(\langle \mathsf{sign}(\langle \mathsf{c2h}, x_{\mathsf{sn}}, \rho_{\mathsf{sn}}, x_{\mathsf{pks}}, x_h\rangle, x_{\mathsf{kb}}), \rho_{\mathsf{sn}}, x_{\mathsf{pks}}, x_h\rangle, \mathsf{state}_1\ ]$

B_2^{go}**:** $[\ \mathsf{state}_1, !\mathsf{Spend}(\rho_{\mathsf{sn}}, z, \langle x_s, x_k\rangle, x_{\mathsf{pks}}), x_{\mathsf{wtn}} = \mathsf{sdec}(x_{\mathsf{ew}}, x_k)\ ]$
$\multimap [\ h(x_k) \approx x_h, f(x_{\mathsf{wtn}}) \approx x_{\mathsf{res}}, \mathsf{Witness}(x_{\mathsf{res}})\ ] \mapsto [\ ]$

B_2^{ab}**:** $[\mathsf{state}_1, !\mathsf{HCoin}(x_{\mathsf{sn}}, \langle g(x_{\mathsf{kb}}), x_{\mathsf{pks}}, x_h\rangle)] \multimap [\ \mathsf{Unspent}(x_{\mathsf{sn}})\ ] \mapsto [\mathsf{Out}(\langle \mathsf{sign}(\langle \mathsf{h2cr}, x_{\mathsf{sn}}, \rho_{\mathsf{sn}}\rangle, x_{\mathsf{kb}}), \rho_{\mathsf{sn}}\rangle)]$

Definition 2. *Let Q be a set of (protocol and ledger) rules and Ψ be a set of restrictions. We say that (Q, Ψ) is a*

- coin infrastructure *if Q produces* $!\mathsf{Spend}(u_{\mathsf{coin}}, \overline{u}, u_{\mathsf{pk}})$ *ledger facts and $\Psi_{\mathsf{coin}} \subseteq \Psi$ (see Figure 1 and Example 6);*
- time infrastructure *if Q produces* $!\mathsf{Time}(u)$ *actions (see Example 7 and Example 8);*
- key infrastructure *if $Q_{\mathsf{keys}} \subseteq Q$ (see Example 4)*
- function model *if Q contains the rules Q_{func}:*

$$[\mathsf{Fr}(x_w)] \Rightarrow [!\mathsf{Witn}(x_w), \mathsf{Out}(f(x_w))]\ ;\quad [\mathsf{Fr}(x_w)] \Rightarrow [!\mathsf{Res}(f(x_w)), \mathsf{Out}(x_w)]$$

If all of these are satisfied we say that (Q, Ψ) is a ZKCP-context.

The fact $!\mathsf{Witn}(x_w)$ from a function model is used by an honest seller to determine a witness, and the adversary (playing the role of the buyer) obtains a desired result $f(x_w)$. The fact $!\mathsf{Res}(f(x_w))$ is used by an honest buyer to determine a desired result, and the adversary (playing the role of the seller) obtains the corresponding witness x_w.

Definition 3. *A* ZKCP Seller specification *is given by a set of protocol rules that contains two special rules:*

sell: $[\ \ldots\ ] \multimap [\ \mathsf{Sell}(t_{\mathsf{pk}}, t_{\mathsf{wtn}})\ ] \mapsto [\ \ldots\ ]$
claim: $[\ \ldots\ ] \multimap [\ \mathsf{Claim}(t_{\mathsf{pk}}, t_{\mathsf{wtn}}, t_{\mathsf{time}}, t_{\mathsf{sn}})\ ] \mapsto [\ \ldots\ ]$

The *sell* rule models the start of a seller session, recording in $\mathsf{Sell}(t_{\mathsf{pk}}, t_{\mathsf{wtn}})$ the seller public key and the witness. The *claim* rule models the seller claiming a coin as payment, producing an action fact $\mathsf{Claim}(t_{\mathsf{pk}}, t_{\mathsf{wtn}}, t_{\mathsf{time}}, t_{\mathsf{sn}})$ where $t_{\mathsf{pk}}, t_{\mathsf{wtn}}$ are as above, t_{time} is timeout constrained data, and t_{sn} the claimed coin. In our case studies, t_{time} is either a sn as in Ex. 7 or a secret key share, cryptographically committed as in Ex. 8. See in Fig. 2 the formal Seller specification for the protocol above.

Definition 4. *Let (Q, Ψ) be a ZKCP-context and $\mathcal{S}$ be a ZKCP Seller specification. We say that these ensure* seller security *if $\mathsf{Q}, \mathcal{S}; \Psi \models \Phi_S$, where Φ_S is defined in Figure 3.*

Fig. 3. Security properties for ZKCP on a ledger

Seller security: witness reveal vs payment: $\Phi_S := \Phi_0 \wedge \Phi_1 \wedge \Phi_2$

$\Phi_0 : \forall x_{\mathsf{pk}}, x_{\mathsf{wtn}}, i, j.\ \mathsf{Sell}(x_{\mathsf{pk}}, x_{\mathsf{wtn}})\ @\ i\ \wedge\ \mathsf{K}(x_{\mathsf{wtn}})\ @\ j \Rightarrow \exists k, y_{\mathsf{pk}}, x_t, x_{\mathsf{coin}}.\ \mathsf{Claim}(y_{\mathsf{pk}}, x_{\mathsf{wtn}}, x_t, x_{\mathsf{coin}})\ @\ k$

$\Phi_1 : \forall \overline{y}, \overline{z}, x.\ \mathsf{Claim}(\overline{y}, x)\ @\ i\ \wedge\ \mathsf{Claim}(\overline{z}, x)\ @\ j \Rightarrow i = j$

$\Phi_2 : \forall x_{\mathsf{pk}}, x_{\mathsf{wtn}}, x_t, x_{\mathsf{coin}}, i, j.\mathsf{Claim}(x_{\mathsf{pk}}, x_{\mathsf{wtn}}, x_t, x_{\mathsf{coin}})\ @\ i \wedge\ !\mathsf{Spend}(x_{\mathsf{coin}}, z, y, z_{\mathsf{pk}})\ @\ j$
$\Rightarrow z_{\mathsf{pk}} = x_{\mathsf{pk}}\ \vee\ \exists k.\ k \leq j\ \wedge\ !\mathsf{Time}(x_t)\ @\ k$

Buyer security: pay gives witness or refund: $\Phi_B := [\ \forall i, j, x_{\mathsf{pk}}, x_{\mathsf{res}}, x_{\mathsf{coin}}, \overline{x}_{\mathsf{state}}.\ (\Phi_0 \wedge \Phi_1)\ ] \wedge \Phi_2$

$\Phi_0(\Psi_0) : \mathsf{Pay}(x_{\mathsf{pk}}, x_{\mathsf{res}}, x_{\mathsf{coin}}, \overline{x}_{\mathsf{state}})\ @\ i \wedge\ !\mathsf{Spend}(x_{\mathsf{coin}}, z, y, z_{\mathsf{pk}})\ @\ j \Rightarrow z_{\mathsf{pk}} = x_{\mathsf{pk}} \vee \Psi_0(y, \overline{x}_{\mathsf{state}})$

$\Phi_1(\Psi_1) : \mathsf{Pay}(x_{\mathsf{pk}}, x_{\mathsf{res}}, x_{\mathsf{coin}}, \overline{x}_{\mathsf{state}})\ @\ i\ \Rightarrow\ \Psi_1(x_{\mathsf{res}}, \overline{x}_{\mathsf{state}})$

$\Phi_2(\Psi_0, \Psi_1) : \forall x_{\mathsf{res}}, y, \overline{x}_{\mathsf{state}}.\ \Psi_0(y, \overline{x}_{\mathsf{state}}) \wedge \Psi_1(x_{\mathsf{res}}, \overline{x}_{\mathsf{state}}) \Rightarrow \exists x_w.\ x_{\mathsf{res}} = f(x_w)\ \wedge\ y, \overline{x}_{\mathsf{state}} \vdash x_w$

Intuitively, the formula $\Phi_S = \Phi_0 \wedge \Phi_1 \wedge \Phi_2$ from Definition 4 ensures that:
• Φ_0: if the other party learns the witness, then (one of) the seller(s) for the corresponding witness is able to claim the payment of a coin into seller's account;
• Φ_1: the other party cannot lead the seller into accepting the same payment twice, e.g. for two different witnesses;
• Φ_2: the payment claimed by the seller will succeed as such on the ledger, unless the corresponding timeout event happened.

Note that, in Φ_0, the key y_{pk} into which payment is claimed is not necessarily equal to the key x_{pk} that engaged in selling the witness: the two keys can differ when there are two sellers for the same witness; then the adversary can learn the witness in one session without paying in the second one. Φ_1 requires care to ensure session specific payments; simply checking unspent conditions on the ledger is not sufficient in case of concurrent sessions. Φ_2 is important because the coin claimed by the seller is jointly constructed with the adversary, so we need to ensure that there is no other way to spend it. The following is proved automatically with Tamarin [35]:

Proposition 1. *For* Seller *of Figure 2,* $\mathsf{Q}_{\mathsf{keys}}, \mathcal{L}_{\mathsf{hash}}, \mathcal{I}_{\mathsf{hash}}, \mathsf{Q}_{\mathsf{func}}, \mathsf{Seller}; \Psi_{\mathsf{coins}} \models \Phi_S$

ZKCP Buyer. As we can see in the $\mathcal{L}_{\mathsf{hash}}$-based protocol presented above, in order to ensure the witness delivery from a ZKCP protocol, the buyer should perform some verification actions on the data (e.g. zero-knowledge proofs) received during the protocol execution. We model these checks by a formula $\Psi_1(x, \overline{x}_{\mathsf{state}})$, where x represents the desired result for the function of interest, and $\overline{x}_{\mathsf{state}}$ represents protocol data that is relevant for buyer's verification actions. Ψ_1 and $\overline{x}_{\mathsf{state}}$ are protocol specific and they are parameters of our definition.

In addition to data received during the protocol execution, the buyer can also rely on data that is published on the ledger, and on the associated constraints that are ensured by the ledger semantics. We model these by $\Psi_0(y, \overline{x}_{\mathsf{state}})$ where y represents the relevant ledger data. For example, in the $\mathcal{L}_{\mathsf{hash}}$-based protocol, the semantics of the ledger ensures that the data y associated to the transaction that spends the hashcoin must contain the preimage of a hash recorded in $\overline{x}_{state}$, if the coin was spent by any party other than the buyer. A part of our security definition will require that Ψ_0 in conjunction with Ψ_1

does indeed reveal the witness. A second part of the definition will require that, if the buyer performed a payment transaction, then the buyer and the ledger will reach a state where Ψ_0 and Ψ_1 hold, or otherwise the buyer can obtain a refund.

Definition 5. *A* ZKCP Buyer specification *is given by a set of protocol rules that contains the special rule* **pay:** $[\ \dots\ ] \multimap [\ \mathsf{Pay}(t_{\mathsf{pk}}, t_{\mathsf{res}}, t_{\mathsf{coin}}, \overline{u}_{\mathsf{state}})\ ] \mapsto [\ \dots\ ]$.

The *pay* rule models the invocation of a payment transaction for a witness, where t_{pk} is the public key of the buyer, t_{res} is the desired result, t_{coin} is the target coin where the buyer makes the payment, and $\overline{u}_{\mathsf{state}}$ is state information that is relevant for obtaining the witness. See Fig. 2 for the Buyer specification in the protocol described above.

Definition 6. *Let (Q, Ψ) be a ZKCP-context and $\mathcal{B}$ be a ZKCP Buyer specification. We say that these ensure* buyer security *if $\mathsf{Q}, \mathcal{B}; \Psi \models \Phi_B$, where Φ_B is defined in Figure 3.*

Intuitively, the formulas Φ_0, Φ_1, Φ_2 from Definition 6 ensures that:
• Φ_0: if the buyer has paid for a witness into a coin, then spending that coin on the ledger will either lead to a refund, i.e. $z_{\mathsf{pk}} = x_{\mathsf{pk}}$, or else the data y associated to the spending transaction together with buyer state data satisfy the constraint Ψ_0;
• Φ_1: before paying, the buyer performs checks entailing the constraint Ψ_1 for the desired result and the buyer state;
• Φ_2: Ψ_0 and Ψ_1 allow to derive a witness for the desired result, by combining transaction data y with data $\overline{x}_{\mathsf{state}}$ gathered from the protocol execution.

Proposition 2. *For* Buyer *from Figure 2 and* $\mathsf{Q} = (\mathsf{Q}_{\mathsf{keys}}, \mathcal{L}_{\mathsf{hash}}, \mathcal{I}_{\mathsf{hash}}, \mathsf{Q}_{\mathsf{func}})$*, we have*

$$\mathsf{Q}, \mathsf{Buyer}; \Psi_{\mathsf{coins}} \models \Phi_B \begin{cases} \overline{x}_{\mathsf{state}} : (x_\pi, x_{\mathsf{ew}}, x_h, x_{\mathsf{pks}}) \\ \Psi_0(y, \overline{x}_{\mathsf{state}}) : \exists y_s, y_h.\ y \approx \langle y_s, y_h \rangle\ \wedge x_h \approx h(y_h) \\ \Psi_1(x_{\mathsf{res}}, \overline{x}_{\mathsf{state}}) : \mathsf{ver}_{\mathsf{zk}}(x_\pi, x_{\mathsf{ew}}, x_{\mathsf{res}}, x_h, x_{\mathsf{pks}}) \approx ok \end{cases}$$

We prove Φ_0 from Φ_B with Tamarin [35]. The properties Φ_1 and Φ_2 are simple local deduction properties that can be checked by hand (if the state of the buyer would be more complex, automated tools can also be used for that).
Observations: • the seller ($\mathcal{S}$) and buyer ($\mathcal{B}$) public keys are linked on the ledger, while this is not a necessary consequence of the security properties. $\mathcal{S}$ does not need to know the public key of $\mathcal{B}$ in advance, while $\mathcal{B}$ does need the public key of $\mathcal{S}$.
• private ledger keys of $\mathcal{S}$ and $\mathcal{B}$ do not have to be secret for security to hold: our models allow corruption of any key by the adversary ($\mathcal{A}$). For $\mathcal{S}$, security follows from the fresh symmetric key created for each session and, for $\mathcal{B}$, from the trusted ledger. Note, however, that these keys allow $\mathcal{A}$ to spend the coins of their owner, but this is independent from the ZKCP protocol. In fact, a basic property of *any* ledger-based protocol should be that it does not reveal secret keys, i.e. $\forall x, i, j.\ !\mathsf{Key}(x) \ @\ i \wedge \mathsf{K}(x) \ @\ j \Rightarrow \exists \ell.\ \ell < j\ \wedge\ \mathsf{Corrupt}(g(x)) \ @\ \ell$. We also prove this property in Tamarin for our models.
• $\mathcal{S}$ cannot reuse the same symmetric key and zero-knowledge proof in two different sessions, even if those sessions are for selling the same witness; • our intruder deduction rules assume a perfect zero-knowledge construction, in particular $\mathcal{A}$ cannot tweak the proof parameters in order to reveal the witness, as exploited by attacks of [16]. In the next section we show that intruder deduction rules can also model finer-grained

properties of cryptographic constructions if required, in particular conditions when the witness may be revealed; • security for $\mathcal{S}$ depends on the timely delivery of transactions to the ledger, while this is not the case for $\mathcal{B}$, who could obtain both the witness and the money back if there was a time delay; • the proof x_π is not necessary for extracting the witness so it can be discarded after verification by $\mathcal{B}$; • our models consider a strong $\mathcal{A}$ and, as such, do not cover the case of weaker, multiple $\mathcal{A}$'s, e.g. for two different buyers that do not collude or do not control the network, but they can be extended to.

5 ZKCP protocol on the basecoin ledger

Managing hashcoins - e.g. applying the hashing algorithm - sets tradeoffs for the agents that maintain the ledger; they may give priority to standard coins, i.e. preferring $\mathcal{L}_{\mathsf{base}}$ over $\mathcal{L}_{\mathsf{hash}}$. Another constraint that needs to be taken into account - by parties engaging in ZKCP - is the complexity of constructing and verifying the zero-knowledge proofs. In this section, we formalize and analyze the protocol of [15], which aims to implement the ZKCP functionality on $\mathcal{L}_{\mathsf{base}}$. Other works, e.g. [18], aim to minimize the zk burden by appealing to special contracts that will be executed only in case of dispute.

Cryptographic primitives. For ZKCP on $\mathcal{L}_{\mathsf{base}}$, [15] adopts timed cryptographic commitments [33, 34], as presented in Example 8, in order to emulate the ledger time-out. To link ledger transitions and data release, [15] exploits algebraic properties of the ECDSA signature used in Bitcoin: relying on homomorphic encryption, e.g. Paillier, an encrypted signature can be constructed from an encryption of the signing key, which can be constructed by adding shares of the signing key on top of an initial encrypted share [36–39]. A Diffie-Hellman group is used to establish a shared key. A special type of zk proof is also needed: a prover can encode the witness and convince the verifier that it can be extracted as soon as some committed structured data - for ZKCP: an ECDSA signature - is revealed. We rely on $\mathcal{I}_{\mathsf{base}}$ from Figure 4 to model these crypto primitives. A term $\mathsf{esign}(m, k, r_1, g(r_1 * r_2), pk(z))$ represents an encrypted partial signature of a message m, with signing key k, randomness share r_1, public randomness $g(r_1 * r_2)$, and encryption public key $pk(z)$. Combining it with the decryption key z and the complementary randomness share r_2, one can compute $\mathsf{sign}(m, k, r_1 * r_2)$. The rules for $\mathsf{extract}$ and $\mathsf{ver}_{\mathsf{zk}}$ model the connection between a valid signature and witness extraction. Time commitments can be checked wrt the public part $g(x)$ of private data x.

Fig. 4. Intruder theory $\mathcal{I}_{\mathsf{base}}$; and $\forall f \in \mathcal{F}^{(k)}.[\mathsf{K}(x_1), \ldots, \mathsf{K}(x_k)] \Rightarrow [\mathsf{K}(f(x_1, \ldots, x_k))]$

$\mathtt{Hom}_{\{\mathtt{g},\mathtt{enc}\}} : [\,\mathsf{K}(g(x)), \mathsf{K}(y)\,] \Rightarrow [\mathsf{K}(g(x * y))] \quad [\,\mathsf{K}(\mathsf{enc}(x, z)), \mathsf{K}(y)\,] \Rightarrow [\mathsf{K}(\mathsf{enc}(x * y, z))]$
$\mathtt{AG} : x * i(x) = 1,\ x * 1 = x,\ x * y = y * x,\ (x * y) * z = x * (y * z)$
$\mathcal{R}_0 : \mathsf{homs}(\mathsf{enc}(k, y), m, r_1, r) \to \mathsf{esign}(m, k, r_1, r, y) \quad \mathsf{dec}(\mathsf{enc}(x, pk(y)), y) \to x$
$\mathsf{decs}(\mathsf{esign}(m, k, r_1, g(r_1 * r_2), pk(z)), r_2, z) \to \mathsf{sign}(m, k, r_1 * r_2)$
$\mathsf{ver}(\mathsf{sign}(x, y, z), x, g(y)) \to \mathsf{ok} \quad \mathsf{open}(\mathsf{com}(x, r), r) \to x \quad \mathsf{extract}(\mathsf{zk}(x, y, z), z) \to x$
$\mathsf{ver}_{\mathsf{tc}}(\mathsf{tcom}(x), g(x)) \to \mathsf{ok} \quad \mathsf{ver}_{\mathsf{zk}}(\mathsf{zk}(x, f(x), \mathsf{sign}(y, z, w)), f(x), y, g(z))) \to \mathsf{ok}$

Jointly signing a message. Assume two parties A_1 (holding k_1, r_1) and A_2 (holding k_2, r_2) want to create $\mathsf{sign}(t, k_1 * k_2, r_1 * r_2)$ for some agreed upon t. Then, say, A_1 can generate a fresh key pair $k, pk(k)$ and send $\mathsf{enc}(k_1, \mathsf{pk}(k))$ to A_2. Relying on $\mathtt{Hom}_{\mathtt{enc}}$, A_2 can obtain $\mathsf{enc}(k_1 * k_2, \mathsf{pk}(k))$, which with $t, r_2, g(r_1 * r_2)$ as arguments to homs gives $\mathsf{esign}(t, k_1 * k_2, r_2, g(r_1 * r_2), \mathsf{pk}(k))$. Sent back to A_1, the joint signature is derived by applying decs to this term and r_1, k. Note that A_1 gets the signature and can decide when to show it to A_2. On the other hand, both parties contribute to randomness in the signature; no party can force a particular value for the randomness. Both of these features will be needed to ensure the security properties for the ZKCP protocol:
1) Based on DH key-exchange and commitments, compute a public key $pk_{12} = g(k_1 * k_2)$ such that the private key $k_1 * k_2$ is secret-shared between the seller ($\mathcal{S}$), who holds $k_1, g(k_2)$, and the buyer ($\mathcal{B}$), who holds $k_2, g(k_1)$. Similarly, secret-shared randomness $r_1 * r_2$ is computed: $\#\mathsf{Public} : pk_{12}, g(r_1 * r_2) \quad \mathsf{Seller} : k_1, r_1 \quad \mathsf{Buyer} : k_2, r_2\#$
2) The key pk_{12} is used for an intermediate transfer from $\mathcal{B}$ to $\mathcal{S}$. The two agree on the transaction that transfers a coin from pk_{12} to $\mathcal{S}$: $\#\mathsf{Public} : t = \langle \mathsf{c2c}, \rho^1_{\mathsf{sn}}, \rho^2_{\mathsf{sn}}, g(\mathsf{ks})\rangle\#$, where $\rho^1_{\mathsf{sn}}, \rho^2_{\mathsf{sn}}$ are fresh public serial numbers and $g(\mathsf{ks})$ is the public key of $\mathcal{S}$. This transaction is not signed, so cannot yet lead to a transfer. Also, $\mathcal{B}$ has not yet transferred coins into pk_{12}.
3) Based on crypto as shown above, $\mathcal{S}$ (with $\mathcal{B}$'s help) obtains $s = sign(t, k_1 * k_2, r_1 * r_2)$. $\mathcal{S}$ checks that s is valid by applying the signature verification algorithm. It then outputs the zero-knowledge proof $\pi = \mathsf{zk}(w, f(w), s)$ and a time commitment to $\mathcal{S}$'s share of the joint secret key: $\#\mathsf{Seller} : s \quad \mathsf{Public} : \pi, tcom(k_1)\#$
4) $\mathcal{B}$ verifies the proof and the time commitment, and transfers a coin to pk_{12}, leading to an update of the ledger: $\#\mathsf{Ledger} : !\mathsf{Coin}(\rho^1_{\mathsf{sn}}, pk_{12})\#$
5) The seller claims ρ^1_{sn} by invoking $\mathsf{R}_{\mathsf{c2c}}$ on the ledger, relying on the signature s obtained previously. The ledger will record a !Spend fact with the corresponding transaction data, including the signature: $\#\mathsf{Ledger} : !\mathsf{Spend}(\rho^1_{\mathsf{sn}}, pk_{12}, s, g(ks))\#$
6) The buyer obtains s from the ledger and extracts the witness from the zk proof: $w = extract(\pi, s)$. If the seller aborted, no one can redeem the coin ρ^1_{sn}, until the time commitment reveals k_1, so the buyer can reconstruct $k_1 * k_2$ and redeem the coin. The formal specification is in Fig. 5, with details of joint signing ommited.

Proposition 3. *For* Seller *and* Buyer *from Figure 5 and* $\mathsf{Q}_{\mathsf{tcom}}$ *from Example 8,*

$$Q, \mathsf{Seller}; \Psi_{\mathsf{coins}} \models \Phi_S \quad Q, \mathsf{Buyer}; \Psi_{\mathsf{coins}} \models \Phi_B \quad Q = (\mathsf{Q}_{\mathsf{keys}}, \mathsf{Q}_{\mathsf{tcom}}, \mathcal{L}_{\mathsf{base}}, \mathcal{I}_{\mathsf{base}}, \mathsf{Q}_{\mathsf{func}})$$
$$\textbf{where } \overline{x}_{\mathsf{state}} : \langle x_\pi, x_{\mathsf{tcom}}, x^{12}_{pk}\rangle, \Psi_0(y, \overline{x}_{\mathsf{state}}) : \exists z, x.\ x_\pi \approx \mathsf{zk}(z, x, x_s) \wedge y \approx x_s;$$
$$\Psi_1(x_{\mathsf{res}}, \overline{x}_{\mathsf{state}}) : \mathsf{ver}_{\mathsf{zk}}(x_\pi, x_{\mathsf{res}}, x_{\mathsf{tcom}}, x^{12}_{pk}) \approx \mathsf{ok}$$

Tamarin verification: we prove Φ_S and Φ_0 for Φ_B automatically with Tamarin relying on the reduction that we present in the next section for termination within 1 minute. We prove two helper lemmas along the way: 1) if the adversary knows a time commitment, then it either knows the committed message at an earlier time, or the commitment is constructed by an honest party; 2) fresh randoms and keys stay secret - unless opened by a time commitment. The Tamarin code is available online [35].
Observations: • as for $\mathcal{L}_{\mathsf{hash}}$, the $\mathcal{S}$ and $\mathcal{B}$ are linked on the ledger; the secret keys of any party can be corrupted, we prove however that the protocol does not itself reveal these keys; • the cryptographic constructions from [15] are a particular instance of $\mathcal{I}_{\mathsf{base}}$;

Fig. 5. ZKCP on $\mathcal{L}_{\mathsf{base}}$; $\mathsf{Seller} = (S_0, \ldots, S_4)$; $\mathsf{Buyer} = (B_0, \ldots, B_3, B_4^{\mathsf{go}}, B_4^{\mathsf{ab}})$

S_0: $[\ !\mathsf{Key}(x_{\mathsf{ks}}), !\mathsf{Witn}(x_{\mathsf{wtn}})\ ] \multimap [\ \mathsf{Sell}(g(x_{\mathsf{ks}}), x_{\mathsf{wtn}})\ ] \mapsto [\ \mathsf{state}_0\ ]$

S_1: $[\ \mathsf{state}_0, \mathsf{Fr}(k_1), \mathsf{Fr}(r_1), \mathsf{Fr}(r)\ ] \Rightarrow [\ \mathsf{Out}(\mathsf{com}(g(k_1), r)), \mathsf{Out}(g(r_1)), \mathsf{state}_1\ ]$

S_2: $[\ \mathsf{state}_1, \mathsf{In}(y_{k_2}), \mathsf{Fr}(k_e)\ ] \Rightarrow [\ \mathsf{Out}(r),\ \mathsf{Out}(\mathsf{enc}(k_1, \mathsf{pk}(k_e))), \mathsf{state}_2\ ]$

S_3: $[\ \mathsf{state}_2 \lceil y_{k_2} = g(x_{k_2}) \rfloor, x_{\mathsf{pk}}^{12} = g(x_{k_2} * k_1), c_k = \mathsf{tcom}(k_1), x_\pi = \mathsf{zk}(x_{\mathsf{wtn}}, f(x_{\mathsf{wtn}}), s)\ ]$
$(\texttt{JointSign} \mapsto t = \langle \mathsf{c2c}, \rho_{\mathsf{sn}}^1, \rho_{\mathsf{sn}}^2, g(x_{\mathsf{ks}}) \rangle, s = \mathsf{sign}(t, \ldots)\ )$ $\Rightarrow [\ \mathsf{Out}(\langle c_k, x_\pi \rangle), \mathsf{state}_3\ ]$

S_4: $[\ \mathsf{state}_3, !\mathsf{Coin}(\rho_{\mathsf{sn}}^1, x_{\mathsf{pk}}^{12})\ ] \multimap [\ \mathsf{Unspent}(\rho_{\mathsf{sn}}^1), \mathsf{Claim}(g(x_{\mathsf{ks}}), x_{\mathsf{wtn}}, k_1, \rho_{\mathsf{sn}}^1)\ ] \mapsto [\ \mathsf{Out}(\langle s, \rho_{\mathsf{sn}}^2, g(x_{\mathsf{ks}}) \rangle)\ ]$

B_0: $[\ !\mathsf{Res}(x_{\mathsf{res}}), !\mathsf{Key}(x_{\mathsf{kb}}), !\mathsf{Pk}(x_{\mathsf{pks}}), !\mathsf{Coin}(x_{\mathsf{sn}}^0, g(x_{\mathsf{kb}}))\ ] \Rightarrow [\ \mathsf{state}_0\ ]$

B_1: $[\ \mathsf{state}_0, \mathsf{In}(\langle x_{ck}, y_{r_1} \rangle), \mathsf{Fr}(k_2), \mathsf{Fr}(r_2)\ ] \Rightarrow [\mathsf{Out}(\langle g(k_2), g(r_2) \rangle), \mathsf{state}_1\ ]$

B_2: $[\ \mathsf{state}_1 \lceil x_{ck} = \mathsf{com}(g(x_{k_1}), x_r), y_{r_1} = g(x_{r_1}) \rfloor, \mathsf{In}(x_r), x_{pk}^{12} = g(x_{k_1} * k_2), x_r^{12} = g(x_{r_1} * r_2)\ ]$
$(\texttt{JointSign} \mapsto t = \langle \mathsf{c2c}, \rho_{\mathsf{sn}}^1, \rho_{\mathsf{sn}}^2, x_{\mathsf{pks}} \rangle, s = \mathsf{sign}(t, \ldots)\ )$ $\Rightarrow [\mathsf{state}_2\ ]$

B_3: $[\ \mathsf{state}_2, \mathsf{In}(\langle x_{\mathsf{tcom}}, x_\pi \rangle), \mathsf{Fr}(r)\ ] \multimap [\ \mathsf{ver}_{\mathsf{zk}}(x_\pi, x_{\mathsf{res}}, t, x_{pk}^{12}) \approx \mathsf{ok}, \mathsf{ver}_{\mathsf{tc}}(x_{\mathsf{tcom}}, g(x_{k_1})) \approx \mathsf{ok},$
$\mathsf{Pay}(g(x_{\mathsf{kb}}), x_{\mathsf{res}}, \rho_{\mathsf{sn}}^1, \langle x_\pi, x_{\mathsf{tcom}}, x_{pk}^{12} \rangle)\ ] \mapsto$
$[\ \mathsf{Out}(\langle \mathsf{sign}(\langle \mathsf{c2c}, x_{\mathsf{sn}}^0, \rho_{\mathsf{sn}}^1, x_{pk}^{12} \rangle, x_{\mathsf{kb}}, r), \rho_{\mathsf{sn}}^1, x_{pk}^{12} \rangle), \mathsf{state}_3\ ]$

B_4^{go}: $[\ \mathsf{state}_3, !\mathsf{Spend}(\rho_{\mathsf{sn}}^1, z, s, x_{\mathsf{pks}}), x_{\mathsf{wtn}} = \mathsf{extract}(x_\pi, s)\ ] \multimap [\ x_{\mathsf{res}} \approx f(x_{\mathsf{wtn}}), \mathsf{Witness}(x_{\mathsf{res}})\ ] \mapsto [\ ]$

B_4^{ab}: $[\ \mathsf{state}_3, !\mathsf{Coin}(\rho_{\mathsf{sn}}^1, g(x_k^{12})), \mathsf{In}(x_{k_1}), \mathsf{Fr}(r), x_k^{12} = x_{k_1} * k_2, x_s = \mathsf{sign}(\langle \rho_{\mathsf{sn}}^1, \rho_{\mathsf{sn}}^2, g(x_{\mathsf{kb}}) \rangle, x_k^{12}, r)\ ]$
$\multimap [\ x_{\mathsf{tcom}} \approx \mathsf{tcom}(x_{k_1}), \mathsf{Unspent}(\rho_{\mathsf{sn}}^1)\ ] \mapsto [\ \mathsf{Out}(\langle x_s, \rho_{\mathsf{sn}}^2, g(x_{\mathsf{kb}}) \rangle)\ ]$

---

it may admit more efficient instances, and our proofs could still be relied on for the security guarantees; • $\mathcal{I}_{\mathsf{base}}$ does not cover the full algebra of homomorphic encryption, where we have $[\ \mathsf{K}(\mathsf{enc}(x, z)), \mathsf{K}(\mathsf{enc}(y, z))\ ] \Rightarrow [\mathsf{K}(\mathsf{enc}(x * y, z))]$. It is however sound when every ciphertext constructed by honest parties uses a fresh key, as in our case study; covering the full theory is a long-standing, still open, problem for protocol verification • the same shared key could be used for the exchange of several witnesses within the timeframe chosen for the time commitment; • contrary to $\mathcal{L}_{\mathsf{hash}}$, the zero-knowledge proof cannot be discarded by $\mathcal{B}$ after verification, since it is necessary for extracting the witness; • on $\mathcal{L}_{\mathsf{hash}}$, $\mathcal{B}$ sets the ledger timeout and $\mathcal{S}$ can accept to proceed; on $\mathcal{L}_{\mathsf{base}}$ it is the other way around with respect to crypto timeout.

6 Homomorphism and abelian group reduction

We take a class of intruder theories that covers the one of Fig. 4; $\mathcal{F}$ contains a set of homomorphic functions $\mathcal{F}_{\mathsf{hom}}$. We reduce any $\mathcal{I}$ from this class to $\mathcal{I}_\Delta$ such that: $\mathcal{I}_\Delta$ is *simpler* than $\mathcal{I}$; $\mathcal{I}_\Delta$ is sound wrt $\mathcal{I}$. First, given any trace τ wrt $\mathcal{I}$, we show that there is $\mathcal{I}_\Delta$ generating τ and where: *(i)* the homomorphic properties are restricted by arguments from honest parties in τ; *(ii)* the abelian group is degenerated, allowing to obtain any factors from products. Second, we augment any set of rules $\mathcal{S}$ to $\mathcal{S}_\Delta$, which records as facts the homomorphic arguments of $\mathcal{S}$, and $\mathcal{I}_\Delta$ is generalized to cover any trace of $\mathcal{S}_\Delta$.

Definition 7. *A* base *for $\mathcal{F}$ is a function Δ with $dom(\Delta) = \mathcal{F}_{hom}$ and $\forall f \in \mathcal{F}_{hom}^{(n)}$. $\Delta(f) \subseteq \mathcal{T}^n$. We assume that Δ is closed modulo* AC*, i.e. $\Delta_f(u * v, \overline{w}) \Rightarrow \Delta_f(v * u, \overline{w})$ and similarly for associativity, and closed by: $\Delta_f(u * v, \overline{w}) \Rightarrow \Delta_f(u, \overline{w})$.*

We extend intruder deduction to rules of the form $[\Delta_f(\overline{x}), M] \Rightarrow [N]$, which have the same semantics as $[M] \Rightarrow [N]$ with the additional constraint that $\overline{x}\theta \in \Delta(f)$ holds for the substitution θ that instantiates the rule.

Definition 8. *We consider the class of intruder theories as defined below (left):*

Initial theory $\mathcal{I}$ (with Hom for all $f \in \mathcal{F}_{hom}$)	Reduced theory $\mathcal{I}_\Delta$ for base Δ
$\mathtt{Hom} : [\ \mathsf{K}(f(x,\overline{z})), \mathsf{K}(y)\ ] \Rightarrow [\mathsf{K}(f(x*y,\overline{z}))]$	$\mathtt{Hom}_\Delta : [\ \Delta_f(x,\overline{z}), \mathsf{K}(y))\ ] \Rightarrow [\mathsf{K}(f(x*y,\overline{z}))]$
$\mathtt{AG} :\ x*i(x) = 1\ ,\ x*1 = x$	$\mathtt{AP} :\ [\ \mathsf{K}(x*y)\ ] \Rightarrow [\mathsf{K}(x)]$
$\quad x*y = y*x\ ,\ (x*y)*z = x*(y*z)$	$\quad x*y = y*x\ ,\ (x*y)*z = x*(y*z)$
$\mathcal{R}_0 : \{l_1 \to r_1, \ldots, l_k \to r_k\}$	$\mathcal{R}_0 : \{l_1 \to r_1, \ldots, l_k \to r_k\}$

We assume that every $l \to r \in \mathcal{R}_0$ satisfies
H1:$top(l), top(r) \notin \mathcal{F}_{hom} \cup \{*, i\}$ ***H2:***$\forall t \in st(r) \setminus st(l).\ top(t) \cap (\mathcal{F}_{hom} \cup \{*, i\}) = \emptyset$
Given such $\mathcal{I}$ and a base Δ, we define the reduced theory $\mathcal{I}_\Delta$ as above (right). $\mathcal{I}$, $\mathcal{I}_\Delta$ also contain the deduction rules $\forall f.[\mathsf{K}(x_1), \ldots, \mathsf{K}(x_k)] \Rightarrow [\mathsf{K}(f(x_1, \ldots, x_k))]$.

H1 and H2 help in proofs [40]; $\mathcal{R}_0$ from Figure 4 respects them. Intuitively, we split the homomorphic argument of f in two parts, e.g. $f(u * v, \overline{w})$, where the factors of v are known by the adversary, while the factors of u are provided by honest parties (in $\mathcal{S}$). When the adversary applies Hom to such a term, to get e.g. $f(u * v * t, \overline{w})$, there is a *smaller* term $f(u, \overline{w})$ that can be used to obtain the same result, since the adversary knows $v * t$. The term u will be added by $\mathcal{S}_\Delta$ to $\Delta(f)$ so $\mathtt{Hom}_\Delta$ can be applied on it.

Proposition 4. *For any $\mathcal{S}$, M_0, M_1 s.t. M_1 can be derived from M_0 using rules in $\mathcal{S} \cup \mathcal{I}$, there is Δ s.t. M_1 can be derived from M_0 using $\mathcal{S} \cup \mathcal{I}_\Delta$; Δ can be iteratively constructed by a set $\mathcal{S}_\Delta$ - augmenting each rule in $\mathcal{S}$ with a constant number of facts.*

Corollary 1. *For any $\mathcal{S}$ and formulas Ψ, Φ, we have $\mathcal{S}_\Delta, \mathcal{I}_\Delta; \Psi \models \Phi \Rightarrow \mathcal{S}, \mathcal{I}; \Psi \models \Phi$*

Scope. The reduction is sound for any set of protocol rules. However, since $\mathcal{I}_\Delta$ allows to freely decompose products, it gives too much power to the adversary (leading to false attacks) for certain classes of protocols, e.g. when a nonce r protects a secret s in $s * r$. The reduction is useful for proofs only when secret data is protected by (homomorphic) cryptographic constructions, e.g. exponentiation, encryption, etc.

7 Related and future work

Several works extend the scope of Tamarin to new cryptographic primitives [41–43] or infrastructure features [44, 45]. Our models contribute to both of these directions. On the crypto side, an open question is to cover deductions like $\mathsf{enc}(u, k), \mathsf{enc}(v, k) \Rightarrow \mathsf{enc}(u * v, k)$, which would allow to model e.g. homomorphic tallying for voting [46]. Protocol verification modulo this theory is studied in [47], where abstractions different from ours are used for reducing the theory, but the case studies are limited to unification problems and relatively simple protocols.

Works complementary to ours aim to provide formal guarantees for code executed on the blockchain [48–50]. Our ledger models are, on one hand, grounded on such guarantees and, on the other hand, they allow to reason about the properties of higher-level

protocols and applications. In future work, we can extend our models to cover more general smart contracts, hybrid ledgers and applications [18,32,51]. Current ZKCP protocols don't allow seller/buyer unlinkability, while the security properties leave scope for it. An open problem is ZKCP on ledgers with more privacy [52–54] and appropriate unlinkability notions.

Acknowledgment

The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC).

References

1. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. Available at `https://bitcoin.org/bitcoin.pdf`.
2. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014. Available at `https://gavwood.com/paper.pdf`.
3. L. M. Goodman. Tezos - a self-amending crypto-ledger, 2014. Available at `https://tezos.com/static/white_paper-2dc8c02267a8fb86bd67a108199441bf.pdf`.
4. Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.
5. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'15)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
6. Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'17)*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.
7. Mohammad Torabi Dashti and Sjouke Mauw. Fair exchange. In Burton Rosenberg, editor, *Handbook of Financial Cryptography and Security.*, pages 109–132. Chapman and Hall/CRC, 2010.
8. N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *Advances in Cryptology - International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '98)*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606. Springer, 1998.
9. Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *Advances in Cryptology - 20th Annual International Cryptology Conference (CRYPTO'00)*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2000.
10. Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *22nd ACM Symposium on Principles of Distributed Computing (PODC'03)*, pages 12–19. ACM, 2003.
11. Yehuda Lindell. Legally-enforceable fairness in secure two-party computation. In *Topics in Cryptology–CT-RSA 2008*, pages 121–137. Springer, 2008.

12. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In *Financial Cryptography and Data Security Workshops (BITCOIN and WAHC'14)*, volume 8438 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 2014.
13. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology - 34th Annual Cryptology Conference (CRYPTO'14)*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439. Springer, 2014.
14. Bitcoin wiki: Zero Knowledge Contingent Payment. `https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment`.
15. Waclaw Banasik, Stefan Dziembowski, and Daniel Malinowski. Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In *21st European Symposium on Research in Computer Security, Part II (ESORICS'16)*, volume 9879 of *Lecture Notes in Computer Science*, pages 261–280. Springer, 2016.
16. Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pages 229–243. ACM, 2017.
17. Steven Goldfeder, Joseph Bonneau, Rosario Gennaro, and Arvind Narayanan. Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin. In *21st International Conference on Financial Cryptography and Data Security (FC'17)*, volume 10322 of *Lecture Notes in Computer Science*, pages 321–339. Springer, 2017.
18. 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.
19. Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium (CSF'16)*, pages 164–178. IEEE Computer Society, 2016.
20. Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *IEEE European Symposium on Security and Privacy (EuroS&P'17)*, pages 451–466. IEEE Computer Society, 2017.
21. Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy (SP'17)*, pages 483–502. IEEE Computer Society, 2017.
22. Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pages 1773–1788. ACM, 2017.
23. Charlie Jacomme and Steve Kremer. An extensive formal analysis of multi-factor authentication protocols. In *31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 1–15. IEEE Computer Society, 2018.
24. 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.
25. Iliano Cervesato, Nancy A. Durgin, John C. Mitchell, Patrick Lincoln, and Andre Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *13th IEEE Computer Security Foundations Workshop, CSFW '00, Cambridge, England, UK, July 3-5, 2000*, pages 35–51. IEEE Computer Society, 2000.
26. 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.

27. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 243–320. MIT Press, 1990.
28. Serge Vaudenay. The security of DSA and ECDSA. In *6th International Workshop on Theory and Practice in Public Key Cryptography (PKC'03)*, volume 2567 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2003.
29. Bitcoin wiki: Hashed Timelock Contracts. `https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts`.
30. Bitcoin wiki: Payment channels. `https://en.bitcoin.it/wiki/Payment_channels`.
31. Bitcoin wiki: Lightning Network. `https://en.bitcoin.it/wiki/Lightning_Network`.
32. Mike Hearn. Corda: A distributed ledger.
33. Ron L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, MIT, Cambridge, MA, USA, 1996.
34. Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology - 20th Annual International Cryptology Conference (CRYPTO'00)*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.
35. Tamarin code for ZKCP protocol verification. `https://www.dropbox.com/sh/ahzbbojm5z0e6a9/AAB6-Pz-RK3xwVznlaqaitfca?dl=0`.
36. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'99)*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.
37. Yehuda Lindell. Fast secure two-party ECDSA signing. In *Advances in Cryptology - 37th Annual International Cryptology Conference (CRYPTO'17)*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644. Springer, 2017.
38. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, pages 1837–1854. ACM, 2018.
39. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, pages 1179–1194. ACM, 2018.
40. Additional material: Tamarin code and long paper version. `https://www.dropbox.com/sh/t74k3q4gxrmo0pw/AADvx0e8WDaZgyf0OQFlElICa?dl=0`.
41. Benedikt Schmidt, Ralf Sasse, Cas Cremers, and David A. Basin. Automated verification of group key agreement protocols. In *IEEE Symposium on Security and Privacy (SP'14)*, pages 179–194, 2014.
42. Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, and Ralf Sasse. Automated unbounded verification of stateful cryptographic protocols with exclusive OR. In *31st IEEE Computer Security Foundations Symposium, CSF'18*, pages 359–373. IEEE Computer Society, 2018.
43. Jannik Dreier, Charles Duménil, Steve Kremer, and Ralf Sasse. Beyond subterm-convergent equational theories in automated verification of stateful protocols. In *6th International Conference on Principles of Security and Trust (POST'17)*, volume 10204 of *Lecture Notes in Computer Science*, pages 117–140. Springer, 2017.
44. Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. *Journal of Computer Security*, 24(5):583–616, 2016.
45. Michael Backes, Jannik Dreier, Steve Kremer, and Robert Künnemann. A novel approach for reasoning about liveness in cryptographic protocols and its application to fair exchange. In *IEEE European Symposium on Security and Privacy (EuroS&P'17)*, pages 76–91. IEEE Computer Society, 2017.

46. Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *20th annual (ACM) symposium on Principles of Distributed Computing (PODC'01)*, pages 274–283. ACM, 2001.
47. Fan Yang, Santiago Escobar, Catherine A. Meadows, José Meseguer, and Paliath Narendran. Theories of homomorphic encryption, unification, and the finite variant property. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, pages 123–133, 2014.
48. Massimo Bartoletti and Roberto Zunino. Bitml: A calculus for bitcoin smart contracts. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, pages 83–100, 2018.
49. Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella Béguelin. Formal verification of smart contracts. In *ACM Workshop on Programming Languages and Analysis for Security (PLAS@CCS'16)*, pages 91–96. ACM, 2016.
50. Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon M. Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, and Grigore Rosu. KEVM: A complete formal semantics of the ethereum virtual machine. In *31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 204–217. IEEE Computer Society, 2018.
51. Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, pages 949–966. ACM, 2018.
52. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy, SP'14*, pages 459–474. IEEE Computer Society, 2014.
53. Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *CoRR*, abs/1506.03471, 2015.
54. Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pages 455–471. ACM, 2017.