Security of the Lightning Network: Model Checking a Stepwise Refinement with TLA$^+$

Matthias Grundmann (ORCID) and Hannes Hartenstein (ORCID)

KASTEL Security Research Labs
Karlsruhe Institute of Technology, Karlsruhe, Germany
{matthias.grundmann,hannes.hartenstein}@kit.edu

Abstract. Payment channel networks such as the Lightning Network are an approach to improve the scalability of blockchain-based cryptocurrencies. The complexity of Lightning, the Lightning Network's protocol, makes it hard to assess whether the protocol is secure. To enable computer-aided security verification of Lightning, we formalize the protocol in TLA$^+$ and formally specify the security property that honest users are guaranteed to retrieve their correct balance. Model checking provides a fully automated verification of the security property, however, the state space of the protocol's specification is so large that model checking is unfeasible. We make model checking of Lightning possible using two refinement steps that we verify using proofs. In a first step, we abstract the model of time and in a second step we use compositional reasoning to separately model check a single payment channel and multi-hop payments. These refinements reduce the state space sufficiently to allow for model checking Lightning with small finite models. Our results indicate that the current specification of Lightning is secure.

Keywords: Model checking · Compositional verification · TLA$^+$ · Payment Channel · Temporal logic · Real-time systems · Bitcoin · Security

1 Introduction

Blockchain-based cryptocurrencies do not scale well with respect to their transaction throughput. One approach to improve said scalability are Payment Channel Networks – a second layer on top of a blockchain that processes payments without writing a transaction for each payment to the blockchain. A *payment channel* between two users is opened by publishing one transaction on the underlying blockchain. Once a payment channel is open, it allows for performing an unlimited number of payments between its two users. Finally, a payment channel is closed by publishing a second transaction. In a payment channel *network*, the participating users are connected by payment channels and can perform multi-hop payments using a path between a payment's sender and recipient over a set of payment channels. The Lightning Network [47] is a payment channel network built on top of Bitcoin [44]. It is being used and experiences a rising adoption [5].

Our goal is to verify that Lightning, the Lightning Network's protocol, is secure, i.e., an honest user finally retrieves on the blockchain the user's correct balance in the payment channel even if other users do not cooperate or are actively malicious. Lightning disincentivizes malicious behavior using a mechanism that allows honest parties to detect and punish malicious behavior. The punishment mechanism as well as Lightning's reliance on time and the number of involved parties make it difficult to assess whether Lightning actually fulfills the security property. Such an assessment might be facilitated by computer-aided methods. In particular, model checking can automatically verify that a protocol fulfills a property and provide a counterexample if the checked property does not hold. Lightning is defined by an official specification [62] that describes all aspects of the protocol and partially the intuition behind the protocol. The specification is not directly usable for a security analysis because the specification contains many implementation details and is not formalized. To enable the use of model checking for Lightning, we contribute a *specification of Lightning* in the formal language TLA$^+$ [33, 34] that formalizes all protocol steps and messages that users send during opening, updating, and closing a payment channel as well as for multi-hop payments. We also contribute a *specification of the security property* of a payment channel network by defining a secure payment system in TLA$^+$. Our security model allows parties to become adversarial. Adversarial parties may omit sending messages or publishing transactions required by the protocol and may publish additional transactions not specified by the protocol.

However, the state space of the specification of Lightning with a model of time and multiple users is so large that model checking is unfeasible. We make model checking of Lightning possible using a stepwise refinement (see Fig. 1). We verify general abstractions with hand-written proofs and use model checking to verify the actual Lightning protocol. We prove that the model of time used in the protocol can be abstracted (① & ③) using ideas from the research of timed automata [3]. This proof generalizes to other explicit real-time specifications in TLA$^+$. In a second step, we prove that it suffices to model check the protocol for single payment channels $\textcircled{2a}$ and the protocol for multi-hop payments ④ separately. These refinements reduce the state space sufficiently to allow for model checking Lightning with the model checker TLC [64]. We use TLC to fully explore the state space of models with payments over up to four hops and with two concurrent payments. To check also larger models as well as the whole stepwise refinement, we use simulation which is a lightweight alternative to model checking where only some random behaviors are explored as opposed to checking the complete state space. While our approach does not give comprehensive formal correctness guarantees, it gives confidence that the specification of Lightning is secure. We leave verifying all refinements with a theorem prover for future work.

We describe Lightning in more detail and give an introduction to TLA$^+$ in Section 2. Related work follows in Section 3. Our approaches for the specification of Lightning and for the specification of the security property are presented in Section 4 and Section 5. In Section 6, we explain our approach for verifying that Lightning fulfills the security property and sketch the ideas behind each abstraction step. We present the results of model checking in Section 7 and discuss limitations of our approach and ideas for future work in Section 8.

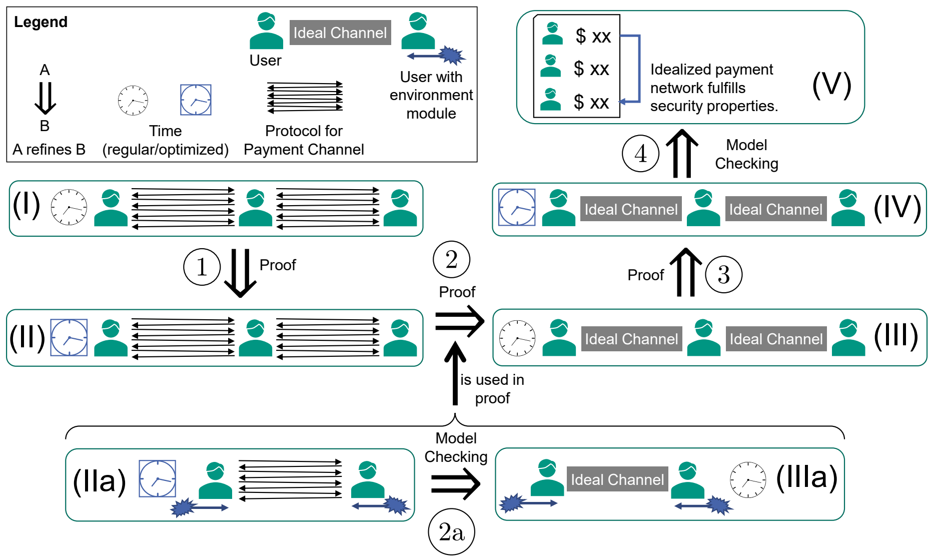

Fig. 1: Structure of the stepwise refinement to show that the Lightning protocol (I) (see Section 4) implements the security property (V) (see Section 5). In ① & ③, we prove that time can be modeled more efficiently. We model check individual payment channels in ⓶ⓐ and interaction of channels in ④. The environment module in spec. (IIa) models the interaction with other channels, ensuring that a channel in spec. (IIa) can be composed with other channels in spec. (II).

2 Fundamentals

In this section, we briefly introduce Lightning and TLA$^+$. For a more detailed introduction, we refer to the extended version of this paper [22].

2.1 Lightning Payment Channel Network

We aim to give an overview of the main ideas of Lightning that contribute to the protocol's complexity. A payment channel is opened by publishing a funding transaction on the blockchain that defines the initial distribution of funds between the two parties. For each payment being made, the two parties agree on a new distribution of funds. Lightning ensures that a user can close a payment channel independently of the other party by asserting that a user always has a valid closing transaction that could be published on the blockchain and distributes the funds according to the latest distribution of funds. After each payment, previously valid closing transactions become outdated but these transactions could still be published on the blockchain. To disincentivize a dishonest user Alice from publishing an outdated closing transaction, the other user, Bob, receives revocation secrets when the closing transaction is outdated. These revocation secrets enable Bob to punish Alice by retrieving not only his assets but also the assets of Alice, if Alice were to publish an outdated closing transaction.

If two users do not have a common payment channel but they are connected over a path of payment channels of other users, they can make multi-hop payments between each other. Intermediate users forward the payment over their channels and receive a small fee for their service. To prevent intermediaries from stealing or losing coins, it must be guaranteed that each intermediary receives an incoming payment on one channel if and only if the intermediary forwards the payment on another channel. Lightning uses Hash Timelocked Contracts (HTLCs) to achieve this atomicity. An HTLC is a contract that encodes the agreement that the recipient receives a specified amount if the recipient proves knowledge of a preimage to a specified hash before a specified time has passed. The recipient of a payment draws a random value x, calculates the hash value $y = H(x)$, and sends y to the sender of the payment. The sender of the payment creates an HTLC with the first intermediary using y as the hash condition for the HTLC. The intermediary creates an HTLC with the next hop and each intermediary repeats this process until the last intermediary creates an HTLC with the recipient of the payment. The recipient fulfills the HTLC by sending x to the last intermediary. Thereby, the payment's recipient receives the payment's amount from the last intermediary. Each intermediary forwards the secret value x back along the route until the sender receives x. Then, all HTLCs are fulfilled and all intermediaries have received and forwarded the payment's amount. The timelocks of the HTLCs are chosen in a descending order from the sender to the recipient, so that each intermediary has enough time to fulfill the incoming HTLC from the previous hop if the next hop fulfills the outgoing HTLC.

2.2 TLA$^+$

We specify Lightning and the security property in TLA$^+$. The Temporal Logic of Actions (TLA) [33] is a temporal logic. The language TLA$^+$ is based on TLA and used to formalize the behavior of a system. TLA$^+$ has been used repeatedly to reason about properties of systems and protocols (see [11,40]). We chose TLA$^+$ because, as a general purpose language, TLA$^+$ allows for specifying arbitrary protocols although abstractions are required for modeling cryptographic primitives (see Section 4) and because there are tools supporting different verification methods from model checking [63,64] to theorem proving [41].

In TLA$^+$, the state of a system is described by a set of variables v. Formally, a state is an assignment of values to variables. The state space of a system is the set of all reachable states. A behavior is a sequence of states. A system is described by defining the set of valid behaviors of the system. A step is a pair of successive states in a behavior. An *action* is a function that maps a step to a boolean value. If an action A maps a step to TRUE, then this step is an A step. An action A is enabled in a state s if there exists an A step starting at state s.

The definition of a system in TLA$^+$ is the set of all valid behaviors of the system. A system is described inductively by a set of initial states and an action that determines valid steps of the system. The set of initial states is defined by a formula *Init* that defines the values that each variable can have in an initial state. An action that commonly has the name *Next* determines which steps are allowed for the system to change its state. By starting in an initial state and performing steps allowed by the action *Next*, the behaviors of the system can be generated. A system with variables v is represented as a formula $Spec = Init \wedge \Box[Next]_v$ where $\Box$ is the *always* operator of temporal logic and $[Next]_v$ means *Next* or a stuttering step in which all variables v are unchanged. An additional conjunct may be a fairness condition, e.g., $WF_v(A)$ which asserts that an A step is taken if the action A is enabled continuously. The *Next* action is typically a disjunction of multiple subactions that define different ways for the system's state to be updated. An action A is described as a conjunction of multiple conjuncts that describe the state in which the action A is enabled and the new state that is reached by an A step. Primed variables (e.g., v') are used to describe the values of the variables in the new state and unprimed variables (e.g., v) describe the values of the variables in the current state.

The variables of a system can be internal or external. From the outside, what a system does is described by the external variables only. A specification S_1 implements (or, equally, refines) specification S_2, noted $S_1 \Rightarrow S_2$, if all external variables of specification S_1 are also external variables of specification S_2 and, when restricting the specifications to these external variables, every behavior of specification S_1 must also be a behavior of specification S_2.

3 Related Work

Aspects of Bitcoin and Lightning were formally analyzed in previous work. Andrychowicz et al. [4] modeled Bitcoin contracts as timed automata and verified them using the UPPAAL model checker [36]. Setzer [54] modeled Bitcoin transactions in Agda [13]. Boyd et al. [14] created a model of a blockchain in Tamarin and analyzed Hash Timelocked Contracts, a primitive that is used by Lightning. Hüttel and Staroveški [27,28] formalized four subprotocols of Lightning and analyzed these protocols using ProVerif for secrecy and authenticity properties. These works on Lightning are complementary to the problem definition in Section 1 as they show lower level properties of subprotocols but not the security of the combined protocol. Rain et al. [16,48] formalized two subprotocols of Lightning and conducted an automated analysis for game-theoretic security. Their work is also complementary to the problem definition above as their formalization of the protocol assumes that an honest party actually can punish a dishonest party. This assumption is a property that we aim to prove. Weintraub et al. [65] analyzed the messages exchanged in Lightning during a single-hop payment. They found an ambiguity in the official specification and show two scenarios in which parties might loose their funds if they do not follow the protocol correctly. In particular, parties may not prematurely consider a payment processed and may not agree out-of-band about a payment's outcome.

The security of Lightning was analyzed before by Kiayias and Thyfronitis Litos [31]. They specified an ideal functionality and used the UC framework [17] to prove that Lightning securely implements this ideal functionality. Compared to our formalization, the protocol definition of [31] considers more details about the cryptographic aspects. While working on our TLA$^+$ formalization of Lightning, we found two subtle flaws in the description of [31] of Lightning that render the formalized protocol insecure. The first flaw concerns an incomplete description of how a user reacts to maliciously published outdated transactions. The second flaw is more subtle and concerns how the data in an input is linked to the spending methods of an output that is spent by this input. A detailed description of the flaws can be found in the appendix and in the extended version of this paper [22]. While we found the first flaw by comparing our formalization to the definitions in [31], we found the second flaw only by model checking when we had a similar flaw in a draft of our formalization. We believe that the specific flaws can be corrected and Lightning actually fulfills the ideal functionality. However, it is difficult and tedious to manually find such flaws in a proof. Using model checking, such issues can be revealed automatically.

Concurrently to our work, Fabiański et al. [21] used the deductive program verification platform Why3 [10] to formalize and verify a simplified variant of Lightning. They also formalized the informal security property that honest users do not loose money. In contrast to our approach of defining the security property by defining the behavior of a secure system, they use a game-based definition which is more complex. While we assume an adversary with limited capabilities, they verified that the formalized protocol is secure even in the presence of arbitrary behavior. Their work shows that a formal verification of even a variant of Lightning considering only single payment channels without HTLCs is a challenging effort.

4 Formalization of Lightning

To verify the security properties of Lightning, we need to formalize Lightning first. In this section, we explain important aspects of our TLA$^+$ formalization of Lightning to show the assumptions and the abstraction layer of the formalization. The complete formalization is available as accompanying artifact [23].

The TLA$^+$ formalization specifies a system with an arbitrary number of users. The behavior of a user is specified as in the official specification [62] with some simplifications that we detail below. The key task of Lightning is to ensure that each user can spend the right transaction output at the right time. Therefore, our model of Lightning focuses on the protocol logic in which users exchange data to build transactions, publish transactions, and observe transactions on the blockchain. To keep the complexity at a manageable level, we do not model fees and we abstract cryptographic primitives like signatures and hash functions.

To ensure that our TLA$^+$ formalization of Lightning captures the behavior of Lightning as closely as possible, the TLA$^+$ formalization follows the structure of the official specification. We make use of the same identifiers for messages as in the official specification, and the states of HTLCs in the formalization can be mapped to those used in Core Lightning [60], an implementation of Lightning.

Lightning uses primitives such as signatures and hash functions that are not directly available in TLA$^+$. For the formalization, we make the perfect cryptography assumption that the adversary cannot break cryptographic primitives, and we use a symbolic representation of cryptographic keys and signatures as done in previous work (e.g., [4,14]). We abstract these primitives by focusing on their relevant properties that are used in Lightning. For example, Lightning is based on the assumption that a hash function is deterministic and easy to evaluate given the preimage but cannot be reverted given a hash value and that two different inputs to a hash function result in two different outputs. In the TLA$^+$ formalization, the preimages that are used for multi-hop payments are not randomly generated but are deterministically assigned based on the associated payment. We model the hash value of a preimage to be equal to the preimage itself and distinguish hashes and preimages by the names of the variables in which a preimage or hash value is stored. In Bitcoin, transaction identifiers are defined as a hash over the transaction. In the formalization, we model transaction identifiers by drawing a new unique value for each transaction when the transaction is created and including that identifier in the transaction.

The TLA$^+$ formalization contains a model of the blockchain and all transaction types used in Lightning defining the conditions how each transaction output can be spent. We model publishing a transaction on the blockchain by a single step that happens instantly, i.e., we assume that users have blockchain connectivity and we make the simplifying assumption that each transaction to be published is included in the next block being created. For the communication between users, we model that messages are delivered reliably and in order but can be arbitrarily delayed.

In Lightning, the height of the blockchain is used as logical time that is relevant for the timeout of HTLCs. We refer to the height of the blockchain as time, which is formalized as a variable that is advanced by integer steps. Thus, our specification of Lightning is a real-time specification. While there are languages especially for modeling real-time systems (e.g., Kronos [15], Uppaal [36]), we use TLA$^+$, a general purpose language. Real-time systems can be modeled in TLA$^+$ using explicit real-time specifications [35] that we define as follows. An *explicit real-time specification* has a set of variables for clocks. Because time is defined in Lightning by the height of the blockchain, we restrict all clocks to have discrete values. Progress of time is modeled by a *Tick* action that advances each clock by the same non-negative integer value and leaves all other variables unchanged. In the specification of Lightning, we model time using a clock representing the height of the blockchain and, for each published transaction, a specific clock which models the time since the transaction's publication and is used to determine whether a timelocked transaction is valid. Some actions in the protocol are urgent (see [12]) meaning that they need to happen before a certain point in time, e.g., a user has to fulfill a HTLC before the HTLC's timeout. We model this by letting each user specify deadlines and not letting time advance beyond a deadline until a step is taken that removes the deadline.

The TLA$^+$ formalization also models adversarial behavior. The adversary model allows the adversary to omit sending messages or publishing transactions. Also, the adversary is allowed to create and publish transactions other than those specified by the protocol. Transactions published by an adversary are only relevant if they spend an output of a transaction that is related to the payment channel. In the formalization, an action models that the adversary publishes transactions in two ways: First, by finding all outputs that are spendable for the adversary and publishing a new transaction that spends these outputs and sends the funds to the adversary. Second, by signing and publishing a transaction that the adversary has already received the other user's signature for (e.g., an outdated commitment transaction). We do not model that the adversary sends messages with arbitrary content because this would significantly increase the specification's complexity. In practice, the effect of the adversary sending messages with arbitrary content is limited because users validate every message they receive. Messages that have an invalid payload or that are received at an invalid point in the protocol execution are ignored. To verify the validation of messages, we explicitly model the validation of every received message and the message's payload, e.g., the validation of signatures and preimages.

We model that any user in the specification can become adversarial. However, we do not allow information exchange between adversarial users which would model a single adversarial entity controlling multiple users. This limitation simplifies the verification of the specification and we consider it future work to extend the specification with a broader adversary model.

5 Security Property

Our goal is to model check the security of Lightning. Our notion of security is captured by the following informal definition. We define a user as being honest if the user behaves as required by the specification of Lightning.

Definition 1 (informal security). *An honest user will finally get paid out on the blockchain at least the user's correct balance.*

This informal definition implicitly concerns four variables: 1. Whether a user is honest. 2. A user's balance in a channel which defines the correct balance that a user expects to have. This balance is affected by the deposited amount and the processed payments. 3. A user's view on whether a payment has been sent or received. 4. A user's balance on the blockchain. To formalize the informal definition, we use TLA$^+$ to define how these four variables are allowed to change by defining the behavior of a secure payment system. The security property is shown in Figs. 2 and 3. The security property has four variables matching the variables described above and is divided into three modules: The module IdealUser describes changes to the variables of a single user, the module IdealPayments ensures that the users' views on which payments have been processed are consistent, and the module IdealPaymentNetwork defines that for all users the specification of the module IdealUser must hold and that the specification of the module IdealPayments must hold. The action *Deposit* describes a deposit as a user's blockchain balance decreasing by an amount and the user's channel balance increasing by the same amount. The action *Withdraw* describes a withdraw by defining that a user's channel balance is reduced by an amount and,

```
MODULE IdealPaymentNetwork
VARIABLES BlockchainBalances, ChannelBalances, Payments, Honest
CONSTANTS UserIds, InitialPayments, Numbers

IdealUser(user) ≜ INSTANCE IdealUser WITH
    UserId ← user,
    ChannelBalance ← ChannelBalances[user],
    BlockchainBalance ← BlockchainBalances[user],
    Payments ← Payments[user],
    Honest ← Honest[user]

IdealPayments ≜ INSTANCE IdealPayments

Spec ≜
    ∧ ∀ user ∈ UserIds : IdealUser(user)!Spec
    ∧ IdealPayments!Spec
```

```
MODULE IdealPayments
EXTENDS Integers
VARIABLE Payments
CONSTANTS UserIds, Numbers

Pay ≜
    ∧ ∀ user ∈ UserIds :
        ∨ UNCHANGED Payments[user]
        ∨ ∃ P ∈ SUBSET {p ∈ Payments[user] : p.state = "NEW"} :
            ∧ ∃ nState ∈ [P → {"ABORTED", "PROCESSED"}] :
                ∧ ∀ p ∈ P :
                    (nState[p] = "PROCESSED" ∧ p.sender = user)
                        ⟹ ∃ rp ∈ Payments'[p.receiver] :
                                rp.id = p.id ∧ rp.state = "PROCESSED"
                ∧ Payments[user]' = (Payments[user] \ P)
                                    ∪ {[p EXCEPT !.state = nState[p]] : p ∈ P}

Spec ≜ Init ∧ □[Pay]_Payments
```

Fig. 2: Formal definition of the security property as a secure payment network. The module *IdealPaymentNetwork* specifies that each user behaves as specified by the module *IdealUser* (see Fig. 3) and that the users' views on which payments have been processed are consistent as specified in the module *IdealPayments* which ensures that a payment can be seen as processed by the payment's sender only if it is seen as processed by the payment's receiver.

```
─────────────────── MODULE IdealUser ───────────────────
EXTENDS Integers, SumAmounts
VARIABLES BlockchainBalance, ChannelBalance, Payments, Honest
CONSTANTS UserIds, UserId, InitialPayments, Numbers
ASSUME Numbers ⊆ Int

Init ≜
    ∧ BlockchainBalance ∈ Numbers
    ∧ ChannelBalance = 0
    ∧ Payments = {pmt ∈ InitialPayments :
                pmt.sender = UserId ∨ pmt.receiver = UserId}
    ∧ Payments ∈ SUBSET [amount : Numbers,
                        sender : UserIds, receiver : UserIds, id : Numbers,
                        state : {"NEW", "ABORTED", "PROCESSED"}]
    ∧ Honest ∈ {TRUE, FALSE}

Deposit ≜
    ∧ ∃ amount ∈ 1 .. BlockchainBalance :
        ∧ BlockchainBalance' = BlockchainBalance − amount
        ∧ ChannelBalance' = ChannelBalance + amount
        ∧ ChannelBalance' ∈ Numbers
    ∧ UNCHANGED ⟨Payments, Honest⟩

Pay ≜
    ∧ ∃ P ∈ SUBSET {pmt ∈ Payments : pmt.state = "NEW"} :
        ∃ nState ∈ [P → {"ABORTED", "PROCESSED"}] :
            ∧ Payments' = (Payments \ P) ∪ {[p EXCEPT !.state = nState[p]] : p ∈ P}
            ∧ LET ProcPmts ≜ {p ∈ P : nState[p] = "PROCESSED"}
                  recAmts ≜ SumAmounts({p ∈ ProcPmts : p.receiver = UserId})
                  sentAmts ≜ SumAmounts({p ∈ ProcPmts : p.sender = UserId})
              IN     ∧ ChannelBalance − sentAmts ≥ 0
                     ∧ ChannelBalance' = ChannelBalance + recAmts − sentAmts
                     ∧ ChannelBalance' ≥ 0
    ∧ UNCHANGED ⟨Honest, BlockchainBalance⟩

Withdraw ≜
    ∧ BlockchainBalance' ∈ Numbers
    ∧ BlockchainBalance' ≥ BlockchainBalance
    ∧ ∃ amount ∈ 0 .. ChannelBalance :
        ∧ ChannelBalance' = ChannelBalance − amount
        ∧ Honest ⟹ BlockchainBalance' ≥ BlockchainBalance + amount
    ∧ UNCHANGED ⟨Payments, Honest⟩

Next ≜ Deposit ∨ Pay ∨ Withdraw
vars ≜ ⟨BlockchainBalance, ChannelBalance, Payments, Honest⟩
Spec ≜
    ∧ Init
    ∧ □[Next]_vars
    ∧ WF_vars(ChannelBalance > 0 ∧ Honest ∧ Withdraw)
─────────────────────────────────────────────────────────
```

Fig. 3: Part of the security property defining how a user's variables may change.

for an honest user, the user's blockchain balance increases by at least the same amount. The action *Pay* of the module IdealUser describes the execution of a set of payments by defining that the sending users' channel balances are decreased by the amounts of payments sent and the receiving users' channel balances are increased by the respective amounts. Payments can be aborted keeping channel balances unchanged. Intuitively, one expects from a secure payment network that the sender of a payment sees the payment as sent (and the payment's balance deducted from the user's channel balance) only if the receiver of the payment sees the payment as received. This condition is enforced by the action *Pay* in the module IdealPayments. The fairness condition of the module IdealUser ensures that the system does not terminate before all honest users have been paid out.

In our formalization of Lightning, the BlockchainBalances variable is refined as the sum of unspent transaction outputs on the blockchain that a user can exclusively spend. In the view of each user, the state of a payment is changed from NEW to PROCESSED when the corresponding HTLC is fulfilled. Because our specification of the protocol allows for adversarial behavior, the result that the protocol specification implements the secure payment system means that no modeled adversarial behavior can break the security property, i.e., the countermeasures implemented in the protocol are sufficient.

6 Verification of Security Properties of Lightning

The state space of the TLA$^+$ specification of Lightning (see Section 4) is too large for model checking. We use stepwise refinement to reduce the specification's state space so that we can model check that it fulfills the security property (see Section 5). In this section, we give an overview of the refinement steps. One reason for the large state space is that there are many equivalent states that only differ by their value of time. In a first abstraction step $\textcircled{1}$ (depicted in Fig. 1), we reduce the number of equivalent states by modeling progress of time more efficiently in specification (II) (see Section 6.1). To further reduce the state space, we abstract the payment channels from being updated by the concrete steps of the Lightning protocol to being updated by idealized steps that merge the effects of multiple protocol steps (see Section 6.2). We model check for a single channel that the protocol steps refine the idealized steps ($\textcircled{2a}$). Based on this result, we prove that specification (II) modeling a network composed of payment channels implements specification (III) modeling a network of idealized payment channels. Because specification (III) uses the original model of time, we again optimize the modeling of time (see Section 6.3). Finally, we model check (see $\textcircled{4}$) that specification (IV) implements the security property defined in specification (V). In this section, we present the ideas behind the proofs. The full proofs can be found in the extended version [22].

6.1 Improved Model of Time

The protocol as specified in specification (I) is too complex for model checking because of the large number of possible states of the protocol. One reason for the huge state space is the modeling of time. There are many bisimilar states that only differ by the value of the clocks. Bisimilarity defines two states s_1 and s_2 to be equivalent if, informally stated, they have the same futures, i.e., for every behavior that starts in state s_1 there exists a matching behavior of steps of the same actions starting at state s_2. Consequently, it suffices to consider only one of the states s_1 and s_2 during model checking. In the area of timed automata [3], this notion of bisimilarity is usually referred to as untimed bisimilarity [2] or time-abstracting bisimilarity [59]. Prior work (e.g. [3, 59]) has proposed to improve model checking of timed automata by grouping all states that are bisimilar in an equivalence class, referred to as a *zone*. A zone graph is constructed by connecting zone z_1 to zone z_2 if zone z_1 contains a state from which a step to a state of zone z_2 exists. During model checking, it suffices to explore the zone graph as a time-optimized specification instead of the possibly much larger state graph of the original specification. To illustrate the effect of the approach, imagine a specification with an initial time value of 1, a single HTLC with timelock 7, and an action with a condition that checks whether the HTLC has timed out. While the original specification would include each state with every possible time value, the time-optimized specification would include only states with time value 1 in which the HTLC has not timed out and time value 7 when the HTLC has reached its timelock. With this approach, the time-optimized specification is bisimilar to the original specification which means that for every behavior in the original specification there exists a behavior in the time-optimized specification *and vice versa*. For the stepwise refinement mapping, we only need the direction that the original specification implements the time-optimized specification. We reduce the state space to an even greater extent by letting the time-optimized specification be a more abstract over-approximation and allowing for behaviors that are not possible in the original specification. More specifically, the zones in the time-optimized specification encode the order in which the outputs of timelocked transactions become spendable. As Lightning does not depend on this order, we reduce the number of zones and allow timelocked transaction outputs to become spendable in any order.

We prove for a general explicit real-time specification that the original specification refines the time-optimized specification by defining a refinement mapping and proving its correctness. The proof can be found in the extended version [22] of this paper. The refinement mapping maps a state s to a state s_{R} that is the zone representative of the respective zone by setting each clock in state s_{R} to the lowest value that the clock can have in the respective zone. The idea of the proof is to show that each step of the original specification starting in state s is mapped to a step of the time-optimized specification starting in state s_{R}. Having proven the optimization in a generalized setting, we prove that the general proof applies to the abstraction from specification (I) to specification (II). Therefore, we explicitly define the zones for specification (I) and prove for every action A and every pair of points in time t_1 and t_2 that if there exists a reachable state so that the action A is enabled at time t_1 but not at time t_2, then the points in time t_1 and t_2 are in different zones.

6.2 Abstraction of Protocol Steps in Payment Channels

Having applied the time optimization, the state space of specification (*II*) is still too large to be explored by model checking. To prune the state space, we divide the model checking problem into two separate refinement steps by specifying an intermediate specification. The intermediate specification specifies idealized channels that abstract from the concrete payment channel protocol. The idealized channels omit protocol details that play a role only for a specific channel and describe only those aspects that are relevant for channels to interact with each other during multi-hop payments, e.g. how the states of HTLCs are updated. Having the intermediate idealized channel specification (*III*), it has to be checked that the protocol specification (*II*) refines the idealized channel specification (*III*) and that the idealized channel specification (*III*) fulfills the security property. Model checking the two refinement steps is a smaller problem than directly checking that specification (*II*) refines the security property for the following reasons: For the second step, the model checker has to explore the states of a specification in which the complex individual channel management has been abstracted. This leads to a much smaller state space. The first step can be checked efficiently because we use ideas from compositional reasoning [1, 19] to check that the channel protocol refines the idealized channels by model checking just a single payment channel. In the following, we explain the idea behind this step. In specification (*II*), payment channels are composed in a network. To separate this network into single channels, we have to consider how channels affect each other. Two channels of the same user can affect each other because they share variables, e.g., a variable for the set of preimages known to the user. If a user learns a preimage in one channel, this preimage becomes also available in all other channels of the user. We explicitly specify how a channel can be affected by the channels in its environment by specifying an environment module that contains an action for each step outside the channel that can affect the channel's variables. We specify a single channel with the environment module in specification (*IIa*). Because specification (*IIa*) contains the environment module and the same module for describing a channel's possible actions as in specification (*II*), every step of a channel in specification (*II*) is also possible in specification (*IIa*). We specify a refinement mapping $\textcircled{2a}$ that maps the state of specification (*IIa*) to a state of specification (*IIIa*) which specifies a single idealized channel. By model checking, we verify that the refinement mapping is correct (see Section 7). We define a refinement mapping $\textcircled{2}$ from the protocol specification (*II*) to specification (*III*) with idealized channels that uses the refinement mapping $\textcircled{2a}$ to map each channel in the protocol specification (*II*) to the corresponding idealized channel in specification (*III*). This refinement mapping is correct because every behavior of a channel in specification (*II*) is described by a behavior of specification (*IIa*) which is mapped to a behavior of an idealized channel by the refinement mapping $\textcircled{2a}$. In the extended version [22], we prove that the environment module describes all steps that can affect a payment channel and we prove that the refinement mapping $\textcircled{2}$ is correct based on the assumption that the refinement mapping $\textcircled{2a}$ is correct which we verify by model checking.

6.3 Refinement of Security Property

Specification (III) is defined as a real-time specification in which time can advance by arbitrary natural numbers. This facilitates the refinement mapping $\textcircled{2}$ from specification (II) to specification (III) because it ensures that every step that advances time in specification (II) is also allowed in specification (III). For efficient model checking, we apply the same optimization for time as used above (see Section 6.1) by defining specification (IV) where equivalent states are grouped. By a proof analogously to the proof of Section 6.1, specification (III) implements specification (IV) and, by transitivity, specification (I) implements specification (IV). Finally, we can model check that specification (IV) using idealized channels implements the security property defined in specification (V).

7 Results of Model Checking

We verify the refinement mappings $\textcircled{2a}$ and $\textcircled{4}$ by model checking and use simulation for additional verification for the manual proof steps $\textcircled{1}$, $\textcircled{2}$ and $\textcircled{3}$.

Table 1: Model checking of refinement mapping $\textcircled{2a}$ from specification (IIa) to specification $(IIIa)$

ID	Model	# States	Runtime
C1	Payment from user A to user B	$\sim 10^5$	~ 3 min
C2	Payment from user A over B to C	$\sim 10^5$	~ 8 min
C3	Payment from user C over A and B to D	$\sim 10^5$	~ 8 min
C4	Two payments: Payment from user A to B and payment from user B over A to C	$\sim 10^7$	~ 8 h
C5	Two concurrent payments from user A to B	$\sim 10^8$	~ 8 wks

To model check the refinement mapping $\textcircled{2a}$ from specification (IIa) to specification $(IIIa)$, we use the model checker TLC that explores all reachable states, calculates the refinement mapping on these states and verifies that the mapped states and steps fulfill specification $(IIIa)$. Specification (IIa) models two users and a payment channel and is parameterized by the information about the context of this payment channel, i.e., the other users in the payment channel network, and the payments to be processed. As there are infinitely many possible ways to parameterize specification (IIa), we only check a small selection of configurations that we deem representative. We model check configurations for five different models that are listed in Table 1. To give an impression, the table also shows the magnitude of the number of distinct states that were explored and the time used by TLC (run on 96 CPU cores). Each model starts with two users (A and B) prepared to open a payment channel and TLC explores all possible behaviors for the two users to open the channel, communicate with users in the environment where applicable, process payments, and close the channel. Each checked behavior ends with the channel being closed and the two users having their funds paid out on the blockchain. The simplest model listed in Table 1 is a payment from user A who funded the channel to the other user. Models C2 and C3 are models in which the channel between users A and B is an intermediate hop on a payment that includes users in the environment. Model C4 models two payments: A payment from user A to user B and a payment that user B sends to user C over user A as an intermediate. There are many more states to explore in model C4 as in the previous models because the two payments can partially interleave: After user B has fulfilled the HTLC for the payment from user A to user B, user B can already start sending the payment to user C while the fulfilled HTLC is removed. Model C5 models two payments from user A to user B which is an even larger model as the two payments can interleave from the beginning. By taking about eight weeks to model check, this model is at the limits of what we can model check in reasonable time.

Table 2: Model checking of refinement mapping ④ from specification (IV) to security property (V)

ID	Model	# States	Runtime
M1	Payment from user A over B to user C	$\sim 10^5$	$\sim$ 1 min
M2	Two payments: Payment from user A over B to C and payment from user C over B to A	$\sim 10^7$	$\sim$ 45 min
M3	Two concurrent payments: Payment from user A over B to C and payment from user A to B	$\sim 10^7$	$\sim$ 1 h
M4	Three payments: Payment from user A over B to C, payment from user B to A, and payment from user B to C	$\sim 10^8$	$\sim$ 13 h
M5	Payment from user A over B and C to user D	$\sim 10^8$	$\sim$ 13 h

The models that we model check to verify the refinement mapping ④ from specification (IV) to specification (V) are listed in Table 2. In all models except the last one, we model three users (A, B, and C) and two payment channels: one channel between user A and user B and the other between user B and user C. In the last model, we model four users (A to D) and three payment channels so that a payment from user A to user D is possible. In all these models, we model check multi-hop payments to check that specification (IV) implements the secure payment network in specification (V).

Model checking of larger models than the models described above becomes impractical. We can partially check larger models by using TLC's simulation mode in which the model checker starts in an initial state and chooses each next state randomly. Recent work has shown that using simulation as a 'lightweight' verification where more rigorous methods are not practical can be successful at finding critical flaws [26]. While we specified the refinement mappings and wrote the proofs, we used simulation to check the abstractions ①, ②, and ④ and the whole stepwise refinement which helped to find flaws in our drafts within few minutes. Further, we used simulation to check the abstractions $\textcircled{2a}$ and $\textcircled{4}$ also for larger models with more users and payments.

8 Discussion and Conclusion

Choice of Formalization Language and Tools Protocol verifiers such as Tamarin [39] and ProVerif [8] have successfully been used for unbounded verification of a number of security protocols [6]. These verifiers support modeling cryptographic primitives, allow for stronger adversary models, and can reason about a protocol's properties without the limitations of finite model checking. These tools have successfully been used to verify subprotocols of Lightning [14,27,28]. However, it is challenging to model natural numbers with addition and subtraction of two variables (see [56, page 35] and [9, page 18]) which is required for modeling blockchain transactions with amounts as in our TLA$^+$ specification. While we had to abstract cryptographic primitives (see Section 4), the generality of TLA$^+$ allowed us to model all relevant aspects of Lightning. We modeled Lightning in TLA$^+$ because TLA$^+$ does not restrict the way properties are proven, whether manually, using an explicit-state [64] or a symbolic model checker [63], or a tool for automated reasoning [41]. We used the explicit-state model checker TLC. The automated model checking process and the generation of counterexamples facilitated our process of defining the intermediate specification (*III*) and the complex refinement mapping from specification (*II*) to specification (*III*). Because we used model checking, we could only verify the security for a number of four users. There might be attacks that only apply when there are more than four users; these attacks would not be discovered by our approach. A further consequence of the choice for model checking was that we had to restrict the adversary model to restrict the messages that an adversary can send. However, the TLA$^+$ specification could also be verified using unbounded verification with a theorem prover [41]. Currently, writing such a proof seems too effortful. However, it will be facilitated by future advancements in assistance and automation for theorem proving.

Limitations and Future Work To formalize Lightning, we left out aspects that are not required for security, e.g., the fees that the sender of a payment pays to intermediate hops, key rotation and onion-routing for increased privacy, and route finding for multi-hop payments. Also, the model of the blockchain could be augmented by considering reorganizations and delays for transaction inclusion. Our adversary model restricts the capabilities of an adversary by disallowing the sending of messages with arbitrary content and the exchange of information between adversarial users. While we had to make these restrictions to keep the specification's state space at a manageable level, follow up work could find optimizations that allow for making the adversary stronger. While this work was in progress, the official Lightning specification was extended to allow both parties to deposit coins into a channel during opening (see [61, Channel Establishment v2]). Our method can be used to formalize and analyze the security of this advancement as well. While we put a focus on a security property, future work could also use the same approach to analyze other properties. For example, it could be shown that, assuming honest and cooperating users and timely delivery of messages, payments are guaranteed to succeed.

Known Attacks on Lightning Prior work has identified several attacks on the assumptions of Lightning and properties that are not included in our security definition. Several works discuss griefing [37,42,46,50] and other denial-of-service attacks [58] in which no funds are stolen but the regular operation is disturbed. In the wormhole attack [38,57], an attacker reroutes a payment and receives the fees intended for other intermediate hops, however, the actual amount of the payment is unconcerned. Extending the TLA$^+$ formalization of Lightning with fees would allow for modeling the wormhole attack. Further, there are attacks on privacy [7, 20, 25, 29, 32, 45, 51, 52] which is a property that is not included in the security definition used in this work. Other works [24, 43, 49, 55] discuss the violation of the assumption that users can timely publish a transaction on the blockchain. In practice, security flaws are also based on implementations not following the specification, e.g., by missing verification checks [53].

Evaluation of Protocol Modifications Besides proving that the formalization of Lightning fulfills the security property, the TLA$^+$ formalization of Lightning can also be used to test proposed modifications of Lightning. To quickly find flaws, it can suffice to model check only a subset of the specification (e.g., a single channel) and check just lower-level invariants. Such an approach can accelerate protocol development by providing a short feedback loop to developers. To evaluate this idea, we introduced flaws by adapting the formalization of Lightning and verified that the introduced flaws are detected by model checking. As an example, we tested whether the, so called, second-stage transactions for HTLCs can be removed by including the conditions of the outputs of HTLC second-stage transactions directly in a commitment transaction's outputs. Verification with the model checker showed within a few minutes that this makes the protocol insecure and, thus, Lightning cannot be simplified in this way.

Connecting the Specification to an Implementation There exist multiple implementations of Lightning. While the TLA$^+$ specification of Lightning is not an executable implementation, it can be used to validate the correctness of existing implementations. Cirstea et al. [18] have recently shown a lightweight way to connect implementations in imperative languages to a TLA$^+$ specification. Their approach is to collect traces of program executions and check these traces against traces described by the corresponding TLA$^+$ specification. Transferring their approach, is an opportunity for follow up work.

Conclusion We have formalized Lightning and a secure payment system that captures the security property of Lightning. Using stepwise refinement, we were able to check the security of Lightning for small models. The approach could enable specifications of other protocols to be model checked. In particular, the abstraction of time can be generalized as well as the approach of separating model checking of local behavior and behavior in a network. Thus, our approach can be a valuable tool to analyze new versions of Lightning or similar protocols.

Acknowledgments. This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs. The authors thank the anonymous reviewers for their helpful comments.

Appendix: On the Formalization of [31]

We found the following two flaws in the formalization of [31]. While these flaws render the formalized protocol insecure, they are easy to fix and it seems that the security proof could work for the corrected protocol. The following references to figures and page numbers refer to [30, version 20220217:205237].

The first flaw concerns the punishment of the publication of an outdated commitment transaction for which the protocol is specified in Fig. 37, lines 21-25 (page 64). A problem arises in the following situation: An HTLC from user Alice to user Bob was off-chain fulfilled and removed. Now, the HTLC's absolute timelock has passed. Alice is malicious and publishes the outdated commitment transaction that commits the HTLC and the associated HTLC timeout transaction. Bob runs the protocol specified in Fig. 37. In line 22, a revocation transaction is created whose inputs spend all outputs of the outdated commitment transaction. In the situation described, this revocation transaction is invalid. Instead of an input referencing the outdated commitment transaction's HTLC output, the revocation transaction must have an input that references the output of the HTLC timeout transaction. While the protocol as formalized in Fig. 37 is incorrect, the security proof on page 90 does not mention the case that a second-stage (timeout or success) HTLC transaction might have been published for an outdated commitment transaction and, thus, the protocol seems correct.

For a scenario that shows the impact of the second flaw, assume that in the payment channel between users Alice and Bob there is an unfulfilled HTLC for a payment from Alice to Bob. After the HTLC's absolute timelock has passed, Bob closes the payment channel by publishing the latest commitment transaction which contains an output o for the HTLC with the spending method $pt_{\mathrm{rev},n+1} \vee (pt_{\mathrm{htlc},n+1}, \texttt{CltvExpiry}\ \text{absolute}) \vee (pt_{\mathrm{htlc},n+1} \wedge ph_{\mathrm{htlc},n+1}, \text{on preimage of } h)$ (see Fig. 40, line 8) where pt (resp. ph) are public keys for which Alice (resp. Bob) has the private key and `CltvExpiry` is the HTLC's absolute timelock. Alice could spend the output o by creating a transaction with an input that uses the disjunct $(pt_{\mathrm{htlc},n+1}, \texttt{CltvExpiry}\ \text{absolute})$. Bob holds the HTLC success transaction that was signed by Alice with the private key for $pt_{\mathrm{htlc},n+1}$ (Fig. 43, line 13). Because the HTLC success transaction, which is meant to spend the third disjunct, also fulfills the conditions of the disjunct $(pt_{\mathrm{htlc},n+1}, \texttt{CltvExpiry}\ \text{absolute})$, Bob could receive the amount of the HTLC without knowing the preimage. One way to correct this problem would be to transform the disjunction in Fig. 40, line 8 into a list of spending methods and add the corresponding indices to the inputs in Fig. 43, line 13. Another way is taken by Lightning which uses the operator `CHECKLOCKTIMEVERIFY` that verifies that a spending transaction has a certain `locktime` set. As Bob's HTLC success transaction has the `locktime` set to 0, the success transaction cannot fulfill the spending method meant for the timeout.

References

1. Abadi, M., Lamport, L.: Conjoining specifications. ACM Transactions on Programming Languages and Systems **17**(3), 507–535 (May 1995). https://doi.org/10.1145/203095.201069
2. Alur, R., Courcoubetis, C., Henzinger, T.A.: The Observational Power of Clocks. In: Jonsson, B., Parrow, J. (eds.) CONCUR '94: Concurrency Theory. pp. 162–177. Springer, Berlin, Heidelberg (1994). https://doi.org/10.1007/978-3-540-48654-1_16
3. Alur, R., Dill, D.: Automata for modeling real-time systems. In: Paterson, M.S. (ed.) Automata, Languages and Programming. pp. 322–335. Springer, Berlin, Heidelberg (1990). https://doi.org/10.1007/BFb0032042
4. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, Ł.: Modeling Bitcoin Contracts by Timed Automata. In: Legay, A., Bozga, M. (eds.) Formal Modeling and Analysis of Timed Systems. pp. 7–22. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-10512-3_2
5. Barbaravičius, V.: Year-over-Year Data Shows Rising Lightning Network Adoption | CoinGate (Aug 2024), https://coingate.com/blog/post/lightning-network-year-over-year-data
6. Basin, D., Cremers, C., Dreier, J., Sasse, R.: Tamarin: Verification of Large-Scale, Real-World, Cryptographic Protocols. IEEE Security & Privacy **20**(3), 24–32 (May 2022). https://doi.org/10.1109/MSEC.2022.3154689
7. Biryukov, A., Naumenko, G., Tikhomirov, S.: Analysis and Probing of Parallel Channels in the Lightning Network. In: Eyal, I., Garay, J. (eds.) Financial Cryptography and Data Security. pp. 337–357. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-18283-9_16
8. Blanchet, B.: Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. Foundations and Trends in Privacy and Security **1**(1-2), 1–135 (Oct 2016). https://doi.org/10.1561/3300000004
9. Blanchet, B.: The Security Protocol Verifier ProVerif and its Horn Clause Resolution Algorithm. Electronic Proceedings in Theoretical Computer Science **373**, 14–22 (Nov 2022). https://doi.org/10.4204/EPTCS.373.2
10. Bobot, F., Filliâtre, J.C., Marché, C., Paskevich, A.: Why3: Shepherd Your Herd of Provers. In: Boogie 2011: First International Workshop on Intermediate Verification Languages. p. 53 (2011). https://doi.org/10/document
11. Bögli, R., Lerena, L., Tsigkanos, C., Kehrer, T.: A Systematic Literature Review on a Decade of Industrial TLA+ Practice. In: Kosmatov, N., Kovács, L. (eds.) Integrated Formal Methods. pp. 24–34. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-76554-4_2
12. Bornot, S., Sifakis, J., Tripakis, S.: Modeling Urgency in Timed Systems. In: de Roever, W.P., Langmaack, H., Pnueli, A. (eds.) Compositionality: The Significant Difference. pp. 103–129. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/3-540-49213-5_5

13. Bove, A., Dybjer, P., Norell, U.: A Brief Overview of Agda – A Functional Language with Dependent Types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics. pp. 73–78. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_6
14. Boyd, C., Gjøsteen, K., Wu, S.: A Blockchain Model in Tamarin and Formal Analysis of Hash Time Lock Contract. In: DROPS-IDN/v2/document/10.4230/OASIcs.FMBC.2020.5. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/OASIcs.FMBC.2020.5
15. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: Ravn, A.P., Rischel, H. (eds.) Formal Techniques in Real-Time and Fault-Tolerant Systems. pp. 298–302. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/BFb0055357
16. Brugger, L.S., Kovács, L., Petkovic Komel, A., Rain, S., Rawson, M.: CheckMate: Automated Game-Theoretic Security Reasoning. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 1407–1421. CCS '23, Association for Computing Machinery, New York, NY, USA (Nov 2023). https://doi.org/10.1145/3576915.3623183
17. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145 (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
18. Cirstea, H., Kuppe, M.A., Loillier, B., Merz, S.: Validating Traces of Distributed Programs Against TLA+ Specifications. In: Madeira, A., Knapp, A. (eds.) Software Engineering and Formal Methods. pp. 126–143. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-77382-2_8
19. Clarke, E., Long, D., McMillan, K.: Compositional model checking. In: [1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science. pp. 353–362 (Jun 1989). https://doi.org/10.1109/LICS.1989.39190
20. van Dam, G., Kadir, R.A., Nohuddin, P.N.E., Zaman, H.B.: Improvements of the Balance Discovery Attack on Lightning Network Payment Channels. In: Hölbl, M., Rannenberg, K., Welzer, T. (eds.) ICT Systems Security and Privacy Protection. pp. 313–323. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58201-2_21
21. Fabiański, G., Stefański, R., Thyfronitis Litos, O.S.: A Formally Verified Lightning Network (Apr 2025), https://fc25.ifca.ai/preproceedings/63.pdf
22. Grundmann, M., Hartenstein, H.: Model Checking the Security of the Lightning Network (May 2025). https://doi.org/10.48550/arXiv.2505.15568
23. Grundmann, M., Hartenstein, H.: TLA+ Specification of Lightning, Security Property, and Refinement Mappings. Zenodo (Sep 2025). https://doi.org/10.5281/zenodo.17206471
24. Harris, J., Zohar, A.: Flood & Loot: A Systemic Attack on The Lightning Network. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 202–213. AFT '20, Association for Computing Machinery, New York, NY, USA (Oct 2020). https://doi.org/10.1145/3419614.3423248
25. Herrera-Joancomartí, J., Navarro-Arribas, G., Ranchal-Pedrosa, A., Pérez-Solà, C., Garcia-Alfaro, J.: On the Difficulty of Hiding the Balance of Lightning Network Channels. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security. pp. 602–612. Asia CCS '19, Association for Computing Machinery, Auckland, New Zealand (Jul 2019). https://doi.org/10.1145/3321705.3329812

26. Howard, H., Kuppe, M.A., Ashton, E., Chamayou, A., Crooks, N.: Smart Casual Verification of the Confidential Consortium Framework. In: Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation. pp. 259–276. USENIX Association, Philadelphia, PA, USA (2025)
27. Hüttel, H., Staroveški, V.: Secrecy and Authenticity Properties of the Lightning Network Protocol. pp. 119–130 (Feb 2020), https://www.scitepress.org/Link.aspx?doi=10.5220/0008974801190130
28. Hüttel, H., Staroveški, V.: Key Agreement in the Lightning Network Protocol. In: Furnell, S., Mori, P., Weippl, E., Camp, O. (eds.) Information Systems Security and Privacy. pp. 139–155. Communications in Computer and Information Science, Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-94900-6_7
29. Kappos, G., Yousaf, H., Piotrowska, A., Kanjalkar, S., Delgado-Segura, S., Miller, A., Meiklejohn, S.: An Empirical Analysis of Privacy in the Lightning Network. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 167–186. Springer, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64322-8_8
30. Kiayias, A., Thyfronitis Litos, O.S.: A Composable Security Treatment of the Lightning Network (2019), https://eprint.iacr.org/2019/778
31. Kiayias, A., Thyfronitis Litos, O.S.: A Composable Security Treatment of the Lightning Network. In: 2020 IEEE 33rd Computer Security Foundations Symposium (CSF). pp. 334–349 (Jun 2020). https://doi.org/10.1109/CSF49147.2020.00031
32. Kumble, S.P., Epema, D., Roos, S.: How Lightning's Routing Diminishes its Anonymity. In: Proceedings of the 16th International Conference on Availability, Reliability and Security. pp. 1–10. ARES '21, Association for Computing Machinery, New York, NY, USA (Aug 2021). https://doi.org/10.1145/3465481.3465761
33. Lamport, L.: The temporal logic of actions. ACM Transactions on Programming Languages and Systems **16**(3), 872–923 (May 1994). https://doi.org/10.1145/177492.177726
34. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA (2002)
35. Lamport, L.: Real-Time Model Checking Is Really Simple. In: Borrione, D., Paul, W. (eds.) Correct Hardware Design and Verification Methods. pp. 162–175. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/11560548_14
36. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer **1**(1-2), 134–152 (Dec 1997). https://doi.org/10.1007/s100090050010
37. Lu, Z., Han, R., Yu, J.: General Congestion Attack on HTLC-Based Payment Channel Networks. In: DROPS-IDN/v2/document/10.4230/OASIcs.Tokenomics.2021.2. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/OASIcs.Tokenomics.2021.2
38. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In: Proceedings 2019 Network and Distributed System Security Symposium. Internet Society, San Diego, CA (2019). https://doi.org/10.14722/ndss.2019.23330

39. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 696–701. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
40. Merz, S.: Formal specification and verification. In: Concurrency: the Works of Leslie Lamport, pp. 103–129. Association for Computing Machinery, New York, NY, USA (Oct 2019), https://doi.org/10.1145/3335772.3335780
41. Microsoft Research – Inria Joint Centre: TLA+ Proof System (2025), https://proofs.tlapl.us/doc/web/content/Home.html
42. Mizrahi, A., Zohar, A.: Congestion Attacks in Payment Channel Networks. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 170–188. Springer, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64331-0_9
43. Nadahalli, T., Khabbazian, M., Wattenhofer, R.: Timelocked Bribing. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 53–72. Springer, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64322-8_3
44. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. Tech. rep. (2008)
45. Ndolo, C., Tschorsch, F.: On the (Not So) Surprising Impact of Multi-Path Payments on Performance And Privacy in the Lightning Network. In: Katsikas, S., Cuppens, F., Cuppens-Boulahia, N., Lambrinoudakis, C., Garcia-Alfaro, J., Navarro-Arribas, G., Nespoli, P., Kalloniatis, C., Mylopoulos, J., Antón, A., Gritzalis, S. (eds.) Computer Security. ESORICS 2023 International Workshops. pp. 411–427. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-54204-6_25
46. Pérez-Solà, C., Ranchal-Pedrosa, A., Herrera-Joancomartí, J., Navarro-Arribas, G., Garcia-Alfaro, J.: LockDown: Balance Availability Attack Against Lightning Network Channels. In: Bonneau, J., Heninger, N. (eds.) Financial Cryptography and Data Security. pp. 245–263. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_14
47. Poon, J., Dryja, T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Tech. rep. (2016)
48. Rain, S., Avarikioti, G., Kovács, L., Maffei, M.: Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. In: 2023 IEEE 36th Computer Security Foundations Symposium (CSF). pp. 107–122 (Jul 2023). https://doi.org/10.1109/CSF57540.2023.00003
49. Riard, A., Naumenko, G.: Time-Dilation Attacks on the Lightning Network. arXiv:2006.01418 [cs] (Jun 2020), http://arxiv.org/abs/2006.01418
50. Rohrer, E., Malliaris, J., Tschorsch, F.: Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW). pp. 347–356 (Jun 2019). https://doi.org/10.1109/EuroSPW.2019.00045
51. Rohrer, E., Tschorsch, F.: Counting Down Thunder: Timing Attacks on Privacy in Payment Channel Networks. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 214–227. AFT '20, Association for Computing Machinery, New York, NY, USA (Oct 2020). https://doi.org/10.1145/3419614.3423262
52. Romiti, M., Victor, F., Moreno-Sanchez, P., Nordholt, P.S., Haslhofer, B., Maffei, M.: Cross-Layer Deanonymization Methods in the Lightning Protocol. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 187–204. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64322-8_9

53. Russell, R.: Full Disclosure: CVE-2019-12998 / CVE-2019-12999 / CVE-2019-13000 (2019), https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-September/002174.html
54. Setzer, A.: Modelling Bitcoin in Agda (Apr 2018). https://doi.org/10.48550/arXiv.1804.06398
55. Sguanci, C., Sidiropoulos, A.: Mass Exit Attacks on the Lightning Network. In: 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–3 (May 2023). https://doi.org/10.1109/ICBC56567.2023.10174926
56. The Tamarin Team: Tamarin-Prover Manual (2024), https://tamarin-prover.com/manual/master/tex/tamarin-manual.pdf
57. Tikhomirov, S., Moreno-Sanchez, P., Maffei, M.: A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 387–396 (Sep 2020). https://doi.org/10.1109/EuroSPW51379.2020.00059
58. Tochner, S., Zohar, A., Schmid, S.: Route Hijacking and DoS in Off-Chain Networks. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 228–240. AFT '20, Association for Computing Machinery, New York, NY, USA (Oct 2020). https://doi.org/10.1145/3419614.3423253
59. Tripakis, S., Yovine, S.: Analysis of Timed Systems Using Time-Abstracting Bisimulations. Formal Methods in System Design **18**(1), 25–68 (Jan 2001). https://doi.org/10.1023/A:1008734703554
60. Various: Core Lightning, htlc_state.h (2018), https://github.com/ElementsProject/lightning/blob/master/common/htlc_state.h
61. Various: BOLT 2: Peer Protocol for Channel Management (2024), https://github.com/lightning/bolts/blob/master/02-peer-protocol.md
62. Various: BOLT: Basis of Lightning Technology (Lightning Network In-Progress Specifications) (2024), https://github.com/lightning/bolts
63. Various: Apalache | The Symbolic Model Checker for TLA+ (2025), https://apalache-mc.org/
64. Various: TLA+ Toolbox (2025), https://github.com/tlaplus/tlaplus
65. Weintraub, B., Kumble, S.P., Nita-Rotaru, C., Roos, S.: Payout Races and Congested Channels: A Formal Analysis of Security in the Lightning Network. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 2562–2576. CCS '24, Association for Computing Machinery, New York, NY, USA (Dec 2024). https://doi.org/10.1145/3658644.3670315