Auntie: Unobservable Contracts from Zerocash and Trusted Execution Environments*

Adrian Cinal[0009−0001−9789−0470]

[1] NASK National Research Institute
[2] Oktawave
adrian.cinal@nask.pl

Abstract. Privacy-oriented cryptocurrencies like Zerocash only support direct payments and not the execution of more complex contracts. Bitcoin and Ethereum, on the other hand, cannot guarantee privacy, and using them for contract execution leaves open questions about the fungibility of proceeds and requires contract designers to take frontrunning countermeasures. This work reconciles the two worlds and develops a lightweight protocol for decentralized execution of complex contracts that (1) is undetectable to the network at large, (2) maintains anonymity of the potentially mutually distrustful counterparties and unlinkability of their repeated interactions, (3) guarantees fair termination, and (4) is immediately resistant to frontrunning and miner bribery attacks. This is achieved by leveraging the confidentiality and anonymity guarantees of Zerocash and the verifiability and flexibility of trusted execution environments.

Keywords: Fairness · Privacy coins · Trusted execution environments

1 Introduction

Cryptocurrencies are uniquely equipped to enforce fairness in multiparty computation (MPC) by financially penalizing offenders (who, *e.g.*, abort the protocol when the outcome would be unfavourable for them) and compensating their victims. With ownership of cryptocurrency deriving from general consensus about the state of a distributed ledger of transactions, various rules and conditions for transfer can be encoded in these transactions and subsequently enforced by the same consensus mechanism, ensuring that any transfer of money is *atomic* with respect to making good on contractual obligations. Whereas an extensive line of research [1, 2, 4, 8, 16, 18, 24, 26] has been devoted to leveraging cryptocurrencies this way, an overlooked aspect has always been the fungibility of the eventual proceeds from such "contracts." Bentov and Kumaresan in [8], when formalizing fair MPC, assume coins to be fungible and "perfectly indistinguishable from each

* This research was supported by the National Centre for Research and Development under the "Next Gen Cloud" project (grant No. KPOD.05.10-IW.10-0003/24) funded in the "Important Project of Common European Interest – Cloud Infrastructure and Services (IPCEI-CIS)" call.

other," which, say, bitcoins clearly are not due to their provenance being a matter of public record [30,31]. In protocols studied therein, such distinguishable coins change hands in making and claiming deposits: "[t]he amount deposited equals the amount claimed" [8], but participants do not get *their* deposits back but instead those of their counterparties. Meanwhile, coins obtained from, *e.g.*, gambling or illegal auctions could be deemed less valuable. While paying premium relative to the nominal value in order to account for future devaluations could address this, the feasibility of this approach is questionable, as quantifying the associated risk is a challenge [30,31], especially in prevailing regulatory landscape with new legislation affecting the perceived value of coins emerging yearly [34]. Transactions used for implementing contracts are also non-standard or otherwise uncommon, as they must encode (at least some of) the rules of the contract to be enforced by consensus; typically, they interact with smart contracts [4,18,20] or use features such as timelocks [1,2,8,16,26], making them clearly distinguishable from regular point-to-point payments. Furthermore, termination of (or any progress in) a decentralized contract is entirely at the discretion of miners who have the final say about when (or whether at all) to include a transaction in the ledger. This raises concerns about frontrunning [16], censorship [37], and the threat of bribery, since miners see the details of a contract *before* it has been settled. Also, historically, conservative Bitcoin miners have been observed to reject transactions with complex scripts indiscriminately [1,26]. Finally, it may be desirable for participants of a contract, say, a purchase of digital commodities, to keep the details of the purchase confidential [12] or even the very fact of it having occurred undetectable to the public. And, if the parties are distrustful of each other, they may further want to keep their identities secret from one another. All the above concerns are addressed successfully in this work.

1.1 Contributions

The paper proposes a hybrid of *trusted execution environments* (TEEs) and the privacy-oriented cryptocurrency Zerocash [6] and develops principles of fair contract design in this setting which, despite leveraging trusted hardware, turn out to be non-trivial. It documents the design of a contract execution engine called Auntie which guarantees that contracts are *unobservable* [33]. This means (1) they are *undetectable*, *i.e.*, the ledger where they have been executed and one where they have not are indistinguishable to outside parties [33], and (2) the parties remain anonymous to each other, namely, they do not learn each other's blockchain identities (payment addresses). Auntie contracts are also (3) *confidential* [5], in that the parties do not, in general, learn each other's inputs and outputs, including changes to their Zerocash balances. In particular, proceeds from such unobservable (and confidential) contracts are indistinguishable from regular payments. Finally, (4) repeated interactions between the same parties are *unlinkable* [33]. Auntie is maximally lightweight and only relies on (authenticated) encryption and cryptography already involved in Zerocash without, *e.g.*, additional zero-knowledge proofs or MPC (compare with, *e.g.*, [9]). With Zerocash only supporting point-to-point payments and no scripts, timelocks, mul-

tisignatures or other gadgets typically involved in decentralized contracts, the key technical challenge becomes formulating principles of contract design that have the TEEs emulate these missing pieces. Finally, applying these principles with existing TEE technology poses another challenge, namely, addressing the security shortcomings of existing TEE solutions. These are also reviewed in this work and their exact impact on privacy and fairness of AUNTIE is assessed. AUNTIE has been implemented, using Zcash [21] and Intel SGX, open-sourced, and tested on the Zcash testnet (see Appendix A).

1.2 Preliminaries

Proof-of-Work Blockchains. Cryptocurrencies work by maintaining a distributed ledger that stores every transaction executed between the participants in the system. This ledger is typically organized as a sequence of bundles of transactions called *blocks* chained together using cryptographic hashes; hence the name *blockchain*. To make rewriting transaction history infeasible, a computational puzzle is associated with each block a solution to which (called a *Proof of Work*) must be published alongside it for the block to be accepted to the ledger. Because "mining" blocks this way takes time (constant in expectation), any valid subchain can serve as a (probabilistic) proof of delay.[3] This is revisited later in the paper. Miners are paid fees by the transaction originators, which are, for clarity of exposition, ignored in this work. Blockchains are expected to satisfy (1) *consistency*, by which all participants converge on a common view of the ledger, (2) *immutability*, by which a transaction once accepted remains in the ledger forever, and (3) *liveness*, by which every valid transaction broadcast to the *mempool* (the staging area of transactions that miners use) is eventually included in the ledger [16]. In practice, liveness is under threat when one accounts for bribing of miners to censor certain valid transactions, an issue addressed in the present work.

Zerocash. Zerocash is a privacy-oriented cryptocurrency introduced by Ben-Sasson *et al.* in [6] and deployed as Zcash [21]. Unlike in, say, Bitcoin, Zerocash transactions reveal neither the participants nor the transacted amounts in plaintext. Instead, miners are convinced that a transaction is well-formed, *i.e.*, spends valid and previously unspent transaction outputs, and its amounts balance, through zero-knowledge proofs. Specifically, when a Zerocash note is minted, a commitment c to it (*i.e.*, to its amount and recipient address A) is added to a global accumulator. Trapdoors for this commitment are encrypted under the recipient's public key and included in the transaction published on the blockchain. To spend this note, one must generate a (non-interactive zero-knowledge) proof of membership of the commitment c in the accumulator and

[3] Difficulty of the puzzle gets adjusted over time to account for changes in the mining power of the network. For simplicity, this work assumes the difficulty is constant for the duration of the execution of a contract.

of spend authority, *i.e.*, knowledge of the secret key a associated with the recipient address A committed to in c. For brevity, this work will talk about simply *signing* the input instead. (In Zcash [21], the input is indeed signed with a randomized key r and the link between r and a is what is then proved in zero knowledge.) To prevent double-spending, a *serial number* is derived deterministically from the note, revealed and proved consistent (again, in zero knowledge) in the transaction spending the note. If a serial number is ever repeated, the offending transaction is rejected by the network. Importantly, to efficient (probabilistic polynomial-time) adversaries, Zerocash transactions leak negligible information about who the participants are or what the transacted amount is. This enables unobservability of contracts instantiated on top of Zerocash and has important implications for their game-theoretic modelling: with parties learning no information (or, more precisely, negligible information) during contract execution, it is often possible to view the interaction as a simultaneous-move game (resistant to frontrunning). This is elaborated in present work. Finally, indistinguishability of transactions [6] makes bribing or otherwise colluding with miners difficult, as it becomes impossible to point to a single transaction that miners should censor for some reason.

Trusted Execution Environments. Trusted execution environments (TEEs) are hardware components that offer a promising alternative to heavyweight cryptography like zero-knowledge proofs, fully homomorphic encryption, or multiparty computation. They enable code to be executed with confidentiality and integrity guarantees even in the presence of a malicious operating system or hypervisor. TEEs typically also support *remote attestation* [32], a procedure through which a remote relying party can be convinced that the correct program is executing on an authentic hardware platform in a correctly established TEE. It involves signing a report produced by the hardware (which includes a measurement of the TEE and related metadata) with an *attestation key* endorsed by the CPU vendor. If the relying party trusts the CPU vendor (which they implicitly do anyway), they may now trust a piece of hardware on a remote (counterparty's) machine. While the technology is still maturing and vulnerabilities are being actively patched, it has already found many applications in the cryptocurrency space [3, 7, 7, 15–17, 19, 20, 22, 24, 27–29, 41].

This work assumes the standard TEE adversary model, where the attacker controls the operating system (that schedules the TEE) as well as the external interfaces of the TEE. While TEEs are supposed to guarantee data confidentiality as well as control flow and memory integrity, any interaction with the outside world, in particular, network and disk access, must be intermediated by the untrusted host. This allows for malicious data to be injected into the TEE and messages to be reordered or dropped altogether. In particular, the TEEs cannot rely on their hosts to give them accurate view of the blockchain, as it may be in the host's best interest to isolate the TEE, by itself having no meaningful sense of time, and schedule it off the CPU for sufficiently long to present it with a forged subchain. (Notably, the host will not be able to forge specific

transactions, as these are digitally signed, but may try convincing the TEE that a transaction was *not* published.) Similarly to prior work [16, 28, 35, 42], this work views vulnerabilities of the underlying TEE technology a subject of separate research orthogonal to the study of protocols built on top of TEEs and assumes, for the most part, that they are ideal, namely, perfect black-box devices [32] that leak no data across their interface via side channels and guarantee perfect control flow and memory integrity resistant to fault injection (*cf.* [22]). This "UC-style" [32] approach is justified by having the design explored in this work be TEE vendor-agnostic. In particular, it can be implemented using potentially next-generation TEEs. The impact of known vulnerabilities, however, should they be exploited, is assessed in Section 4 which details which fairness and privacy guarantees of Auntie are upheld if physical protections fail. Notably, as Section 4 elaborates, participants in Auntie cannot lose money even in the presence of unbounded side-channel leakage (*cf.* [38]) in contrast to, *e.g.*, [16].

2 Related Work

Most relevant for this work is that of Das *et al.* [16], who propose FastKitten: a TEE-based contract execution framework where a single *operator* hosts a TEE and is held accountable by a timelocked *penalty transaction*. The requirement for timelocks support prevents FastKitten from being applied *as is* to a privacy-oriented cryptocurrency like Zerocash.[4] Present work, however, can be viewed as extending FastKitten to the setting of Zerocash by, among other things, replacing penalty transactions with the "encumbrance" of keys within TEEs [3]. Bowe *et al.* in [9] propose Zexe: a contract execution framework extending Zerocash that hides both inputs to a contract and the functionality implemented by it using zero-knowledge arguments. Present work can be viewed as proposing and evaluating a significantly more lightweight alternative to Zexe that can additionally be immediately deployed as an overlay on top of the existing Zerocash ledger [21]. Given that established systems like Zerocash have associated with them strong inertia and network effects, phasing them out in favour of a contract-capable privacy coin does not seem feasible (*cf.* [16]), making immediate applicability an important advantage of Auntie over Zexe. Castejon-Molina *et al.* in [12] construct contingent payments that are unlinkable by virtue of relying on coin mixers. Present work offers much stronger unlinkability due to reliance on Zerocash. It also replaces external intermediaries with TEEs hosted directly by the parties to a contract. Thyagarajan *et al.* in [36] argue the importance of preserving fungibility formalized as indistinguishability of contract execution from regular point-to-point payments. Wüst *et al.* in [42] are the first, to the best of the author's knowledge, to propose using TEEs in conjunction with Zerocash but

[4] Notably, Zcash [21] does (as a fork of Bitcoin) have limited support for timelocks (enough to port FastKitten). To achieve unobservability (and hence fungibility), however, it is important they not be used. This paper shows how fair contract execution can be achieved in the complete absence of timelock facilities.

do it only to offload blockchain monitoring and enable lightweight clients. Privacy of contracts settled on blockchains is further considered in [4, 11, 12, 23, 25]. Full unobservability is outside the scope of all of them. Finally, the synergy between TEEs and blockchains is also explored in a number of other works, including [3, 7, 7, 15, 17, 19, 20, 22, 24, 27–29], and systematized in [41].

3 Unobservable Contracts

For the purposes of this work, a *contract* is a protocol realizing a (randomized) functionality $\mathcal{F}$ that takes, from each *player* $i = 1, \ldots, n$, an input $x_i \in \{0,1\}^*$ as well as a *deposit* of $d_i \in \mathbb{N}$ coins and outputs to each player a string $y_i \in \{0,1\}^*$ and pays them $p_i \in \mathbb{N}$ coins. Assume each player hosts a TEE. Furthermore, as in FastKitten [16], assume there exists a distinguished party, called the *operator*, who also hosts a TEE and that will *clear* the contract. Specifically, the operator's TEE will evaluate $\mathcal{F}$ and create a settlement transaction t that will make the payments p_i accordingly. This operator can be one of the players, just as in FastKitten, or can be an external party paid for their service by the players in fees [16]. For clarity of exposition, contracts are assumed to be stateless, which includes applications of practical interest such as auctions, lotteries and fair exchange (*cf.* [16,18]). Extending Auntie to stateful contracts is straightforward and sketched below.[5]

Suppose the TEEs of all parties (*i.e.*, the players and the operator) have attested in front of all remaining parties to the code they are executing. This includes the functionality $\mathcal{F}$, refund logic, *etc.*. Suppose further that every player i's TEE has also established a secure (*i.e.*, encrypted and authenticated) channel to the operator's TEE with the associated shared key k_i. Notably, the keys k_i are known only to the TEEs and *not* their hosts. No external trusted or semi-trusted parties are assumed such as reliable time servers [14] or arbiters for dispute resolution [18], although, for the purposes of remote attestation of TEEs, access to some public-key infrastructure of the CPU vendor may be needed (*cf.* [32]). We now briefly go through the Auntie protocol at a high level. Details are elaborated in Section 3.2.

We will have each player i's TEE generate a *deposit address* A_i^{dep} with the associated spending key a_i^{dep}. The operator's TEE will also establish a deposit address A_0^{dep} with secret spending key a_0^{dep}. The parties will make their deposits (d_is for the players[6] and some *collateral* Δ for the operator) to the respective deposit addresses A_i^{dep} established by the TEEs they host. Each party (player or operator) will confirm that the remaining parties have published their deposits. The operator's TEE will then evaluate the functionality $\mathcal{F}$, escrow the results y_i with the players' TEEs, and issue a settlement transaction t to its host that

[5] Note that, while payment channels do not satisfy the above definition of a contract, the principles developed in this paper can naturally be leveraged to bestow unobservability on the construction of [28] as well.

[6] Plus extra fees to compensate the operator if needed [16].

effectuates the payments p_i and returns the operator their collateral Δ. If the operator publishes t to the blockchain, players will be able to use it to convince their TEEs that the contract was settled and they should release the outputs y_i.

If the contract gets aborted prematurely, the keys a_i^{dep} will be released to the parties effectively refunding them. To prove to a TEE that the contract was aborted, its host must present a chain of blocks of some length τ extending a (hardcoded) *checkpoint block* C [16]. The TEE can verify the solutions to the Proof-of-Work puzzles as well as that the blocks are chained correctly and not contain t to be convinced the contract was not settled in time (*cf.* [15, 24]). The Proof-of-Work puzzles function as verifiable (probabilistic) delay with the extra property that an honest player does not need to invest any computational resources to produce the proof: it is instead a by-product of the miners maintaining the blockchain and demonstrates that *at least* the time needed by honest miners to produce τ blocks has elapsed.[7] This approach, importantly, does *not* protect against player i's selfish mining: given enough time, i can and *will* produce a subchain of length τ, all while their TEE is isolated from the others. This is fine if the settlement transaction t gets published asynchronously while i is selfishly mining the fraudulent subchain: the key a_i^{dep} eventually released by i's TEE will be worthless as it will only allow attempting double-spending relative to t.

3.1 Security Requirements

This work adopts the notion of fairness from [16]. A contract is *fair* if, whenever there is at least one honest party and except with negligible probability, the contract either (1) terminates correctly, (2) gets aborted before the settlement transaction t has been issued with all honest parties staying financially neutral and no outputs y_i being revealed to any player, or (3) gets aborted after t has been issued with all honest players staying financially neutral, and the dishonest operator losing money. Note how the fairness definition above subsumes the notion of *operator balance security* from [16], which requires that an operator acting honestly cannot lose their collateral.

Say a contract is *undetectable* [33] if no efficient distinguisher, themselves not a party to the contract, can tell apart two ledgers: one on which the contract was executed and one on which it was not (*cf.* the L-IND experiment in the full version of [6]). A contract is *unobservable* [33] if it is undetectable, and each player i only learns negligible information about the ledger identities (addresses) of other players. It is furthermore *unlinkable* [33] if two repeated interactions between the same parties cannot be linked together except with negligible advantage and *confidential* [5] if parties learn negligible information about each other's deposits d_i, payments p_i, inputs x_i, and outputs y_i in the course of its execution (except what is revealed by the evaluation of $\mathcal{F}$ itself). Furthermore, we require that the outcome of $\mathcal{F}$ only be revealed to the parties once the settlement transaction t has

[7] This assumes that no coalition of players can ever outpace honest miners. However, one must assume this anyway to discount the threat of blockchain reorganization and for the notion of fairness (or, indeed, even paying anyone) to be meaningful.

been confirmed on the blockchain or the contract has already been aborted. This guarantees that parties cannot "back out" of the contract when unhappy with the result, *e.g.*, by bribing miners to censor t long enough to claim contract timeout. This last property, which we shall call *strong fairness*, captures a more intuitive notion of fairness that has been overlooked so far in the literature. Otherwise, the fairness guarantees of Auntie are the same as those of FastKitten [16].

Note how, say, in a blind auction setting, true bid secrecy cannot really be achieved without using a cryptocurrency like Zerocash that hides transaction amounts: in previous designs [16], even if the exact bid was not revealed (*e.g.*, it was hidden by a cryptographic commitment or known only to a TEE), an upper bound on it was nevertheless known, as this was what a bidder had deposited in the (smart) contract. This crude frontrunning countermeasure also meant that parties had to overdeposit and lock up capital only to hide their bidding strategy. In this paper, no funds need to be locked by the parties except the actual bids $b_i = d_i$. Finally, in FastKitten [16], it was also strictly necessary that the operator's collateral Δ satisfied $\Delta \geq \sum_{i=1}^{n} d_i$, as any refunds to the players were drawn against the operator's collateral. In Auntie, Δ is a free parameter.

The Auntie protocol developed in the remainder of this work will lead to the following result:

Theorem 1. *Let $\mathcal{F}$ be a functionality as above. Assuming trusted execution environments realizing the TEE ideal functionality [16, 32],* IND-CCA*-secure encryption* (Gen, Enc, Dec)*, and a blockchain realizing the Zerocash ideal functionality, there exists a fair, unobservable, unlinkable, and confidential contract for $\mathcal{F}$.*

Explicit definition of the Zerocash ideal functionality is omitted, as it is exactly the same as the BC ideal functionality in [16], except transactions do not reveal their inputs or outputs thereby accounting for the privacy properties of Zerocash, namely, *ledger indistinguishability* [6]. The TEE ideal functionality was formalized by Pass *et al.* in [32].

Anonymous Attestation. While, in prior work [3, 7, 14–17, 19, 20, 24, 27–29], the remote attestation process was taken for granted, it becomes of concern when striving for full anonymity and, in particular, unlinkability. Indeed, remote attestation involves signing a hardware-generated report with a key that is derived from a hardware root of trust and certified by the CPU vendor, *e.g.*, Intel. So as to prevent linking interactions with the same CPU (and hence, likely, the same counterparty), group signatures can be used for attestation as originally done by Intel [10] (before the cloud, Intel SGX was intended for client devices to, *e.g.*, enforce DRM policies; hence, anonymity of end users was a priority), or the CPU can be reprovisioned with a new attestation key for each contract.[8] Anonymous attestation is the default in formal (UC) modelling of TEEs, in particular, in the definition of the TEE ideal functionality we use [32].

[8] See, *e.g.*, `https://cc-enabling.trustedservices.intel.com/intel-tdx-enabling-guide/02/infrastructure_setup` (accessed: 6 September 2026).

3.2 The Auntie Protocol

An Auntie execution is parametrized by the number of players n, the functionality $\mathcal{F}$, the checkpoint block C relative to which timeouts are proved, the collateral amount Δ, and two periods (expressed in a number of blocks) τ and τ'. These parameters are hardcoded in the TEE programs that are loaded by the parties to their TEEs. The TEEs attest to this in front of all parties. Encrypted and authenticated channels are also established at this point (tied to the attestation evidence) between the TEEs of the players and the TEE of the operator. Write k_i for the key associated with the channel between the TEEs of the operator and of player i. Finally, write h_C for the height of the checkpoint block C.

The pseudocode for the players' and the operator's TEEs is presented in Figures 1 and 2. Maintenance of a state machine that enforces the TEE interface is not abused and calls are made in proper order is omitted for clarity of exposition, but calls to the $\mathsf{Player}(i)$ TEE can only proceed in the following order:

$$\mathsf{Initialize} \to \mathsf{DepositAndInput} \to \mathsf{GetDeposits} \to \mathsf{ConfirmDeposits} \to \mathsf{Settle}\,,$$

whereas calls to the Operator TEE must go as:

$$\mathsf{Initialize} \to \mathsf{ClearContract} \to \mathsf{Finalize}\,.$$

The Refund methods can be called at any point for both types of TEEs.

A number of subroutines are used. CreateWallet creates a Zerocash *wallet* (a, A), *i.e.*, a pair comprising an address A and the associated spending key a. DepositedAmount, on input a transaction t and spending key a, sums the amounts that t credits to the corresponding address A. CreateTx, on input a set of transactions (UTXOs) and output descriptions of the form $\{(p_j, A_j)\}_j$, creates a (yet unauthorized) transaction that spends its inputs and, for each j, credits address A_j with p_j coins. AuthTx attaches signatures to a yet unauthorized transaction. BlocksSinceCheckpoint, on input a chain of blocks B, verifies the blocks are all chained correctly, have valid Proof-of-Work puzzle solutions, and are built on top of the checkpoint block C. It then returns the length of B. AuthdAndBuried, on input an unauthorized settlement transaction $\hat{t}$, spending key a, and a chain of blocks B, verifies the blocks are all chained correctly, have valid Proof-of-Work puzzle solutions, are built on top of the checkpoint block C, and, in particular, contain a transaction t that is the "authorized version" of $\hat{t}$ and includes a signature under a. It then returns the number of blocks in B that have been appended to the block containing t. Both BlocksSinceCheckpoint and AuthdAndBuried return -1 when their inputs are invalid. Failed assertions are assumed to terminate the call with a return value of $\perp$. The pseudocode neglects operator's fees which are straightforward to account for.

A formal (and concise) description of the Auntie protocol is relegated to Appendix B. This section will instead give a more in-depth treatment of the different design decisions and the technical challenges that they addressed.

Contract execution begins with each player i's call to $\mathsf{Player}(i).\mathsf{Initialize}$, which establishes two wallets. The deposit wallet $(a_i^{\mathsf{dep}}, A_i^{\mathsf{dep}})$ will be used to escrow the

Player(i).Initialize

1 : $(a_i^{\mathsf{dep}}, A_i^{\mathsf{dep}}) \leftarrow\!\!\$\ \mathsf{CreateWallet}()$
2 : $(a_i^{\mathsf{pay}}, A_i^{\mathsf{pay}}) \leftarrow\!\!\$\ \mathsf{CreateWallet}()$
3 : **return** A_i^{dep}

Player(i).DepositAndInput(t_i, x_i)

1 : $d_i \leftarrow \mathsf{DepositedAmount}(t_i, a_i^{\mathsf{dep}})$
2 : **return** $\mathsf{Enc}_{k_i}(t_i, x_i, d_i, A_i^{\mathsf{pay}})$

Player(i).GetDeposits(c)

1 : $(\hat{t}, y_i, t_0, t_1, \ldots, t_n) \leftarrow \mathsf{Dec}_{k_i}(c)$
2 : **return** $(t_0, t_1, \ldots, t_n)$

Player(i).ConfirmDeposits(b)

1 : $\sigma_i \leftarrow\!\!\$\ \mathsf{Sign}_{a_i^{\mathsf{dep}}}(\hat{t})$
2 : **return** $\mathsf{Enc}_{k_i}(b, \sigma_i)$

Player(i).Settle(B)

1 : **assert**$(\mathsf{AuthdAndBuried}(\hat{t}, a_i^{\mathsf{dep}}, B) \geq \tau')$
2 : **return** $(a_i^{\mathsf{pay}}, y_i)$

Player(i).Refund(B)

1 : **assert**$(\mathsf{BlocksSinceCheckpoint}(B) \geq \tau)$
2 : **return** a_i^{dep}

Fig. 1: AUNTIE pseudocode for the player's TEE

deposit d_i, whereas the payout wallet $(a_i^{\mathsf{pay}}, A_i^{\mathsf{pay}})$ serves to satisfy the strong fairness property from Section 3.1. If the settlement transaction t were to credit i's address directly, i could identify t already in the mempool, *i.e.*, before it has been confirmed on the blockchain. They could then learn their their (prospective) payout and infer something about the outcome of $\mathcal{F}$. If unhappy with this outcome, they could still bribe miners to censor t long enough to claim timeout and unfairly back out of the contract. More importantly, players colluding with the malicious operator could infer the outcome as well and, if unhappy, have the operator withhold t. While the operator would be punished financially in this case (they would lose the collateral Δ), we still do not want this decision to withhold t to be in any way informed. In game-theoretic terms, the game where the operator chooses to abort or not should be a simultaneous one. To this end, the settlement transaction t will instead credit A_i^{pay}, and player i will only receive the corresponding key a_i^{pay} after t has been confirmed τ' times on the blockchain (*cf.* key encumbrance in [3]). The operator concurrently calls Operator.Initialize to likewise establish a deposit wallet $(a_0^{\mathsf{dep}}, A_0^{\mathsf{dep}})$.

Each player i creates and broadcasts a transaction t_i that credits address A_i^{dep} with d_i coins. Player i inputs both the deposit transaction t_i and the functionality input x_i to their TEE via Player(i).DepositAndInput. The TEE can verify that t_i credits the correct address, namely, A_i^{dep}. It cannot, however, know if t_i was actually published on the blockchain or not. For this, it must rely on i's counterparties. It sends (via the respective untrusted hosts, *i.e.*, player i and the operator) a tuple $(t_i, x_i, d_i, A_i^{\mathsf{pay}})$ to the operator's TEE. The tuple is encrypted under k_i that is known only to player i's TEE and the operator's TEE.

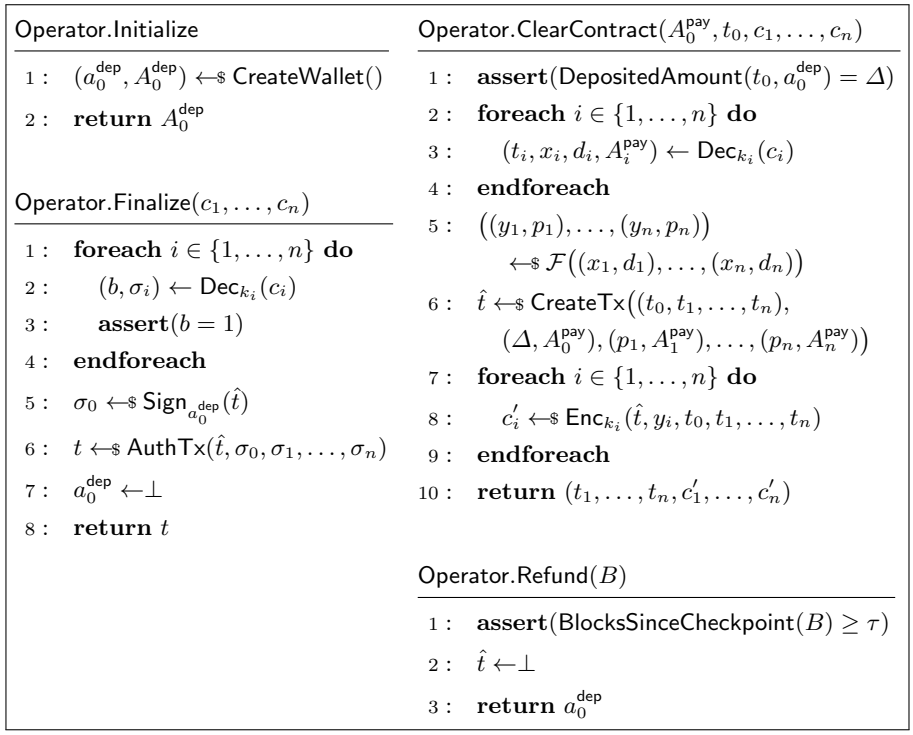

Fig. 2: AUNTIE pseudocode for the operator's TEE

The operator, in the meantime, creates and broadcasts a collateral transaction t_0 that credits A_0^{dep} the collateral amount Δ. They then make a call to the Operator.ClearContract interface with their own (externally created) address A_0^{pay}, the collateral transaction t_0, and the ciphertexts c_i collected from the player's TEEs. The TEE decrypts each c_i to learn the deposit transaction t_i, its amount d_i, the input x_i, and the payout address A_i^{pay}. (Any decryption errors abort the contract.) The amounts d_i and the inputs x_i are then used to evaluate the functionality $\mathcal{F}$. The yet unauthorized (*i.e.*, without the necessary signatures) settlement transaction $\hat{t}$ is created (formatted) at this point and sent to each player i's TEE. (In practice, it is a hash of $\hat{t}$ that is sent here, as that is what gets signed.) The TEEs will sign it in a *CoinJoin*-like fashion, where each player's TEE authorizes their corresponding input to be spent. The deposit transactions t_i including the operator's collateral transaction t_0 are also sent together with the outputs y_i to the players' TEEs at this point.

The operator is given all the deposit transactions t_i by their TEE to verify they have been broadcast to the network and mined. Recall that Zerocash transactions reveal no (non-negligible amount of) information about the amounts deposited or who originated them, so they can be safely handed to the operator to look for them on the blockchain. Similarly, player i is given t_0 and the

deposit transactions t_j of the remaining players by their TEE (via a call to $\mathsf{Player}(i).\mathsf{GetDeposits}$) to confirm they all have been added to the blockchain. If they have, player i calls $\mathsf{Player}(i).\mathsf{ConfirmDeposits}$ with $b = 1$ which results in their TEE's signing $\hat{t}$ and sending the signature to the operator's TEE.

Up until this point, any foul play results in the protocol's halting with all parties eventually being able to redeem their deposits by proving contract timeout to their TEEs (see $\mathsf{Player}(i).\mathsf{Refund}$ and $\mathsf{Operator}.\mathsf{Refund}$ in Figures 1 and 2, respectively). Also, the steps so far could be repeated multiple times to support stateful contracts.

If happy with the deposits $t_1, \ldots, t_n$, the operator calls $\mathsf{Operator}.\mathsf{Finalize}$ with the (encrypted) signatures on $\hat{t}$ received from the players' TEEs. Past this point, the operator's TEE will no longer accept claims of timeout, *i.e.*, the operator cannot back out of the contract now (note that a_0^{dep} is overwritten with $\perp$ in Figure 2). If they default and refuse to broadcast t, the collateral Δ is lost (potentially together with any fees paid by the players to compensate the operator for their service, see [16]), thus punishing the cheating operator. Otherwise, it is paid back to them in t.

Players wait for t to be published and periodically input the suffix of the blockchain, starting from the checkpoint block C, to their TEEs (via a call to the $\mathsf{Player}(i).\mathsf{Settle}$ interface). At some point, t will have been mined and buried under τ' blocks, at which point the TEEs will be convinced the contract has been settled and release the outputs y_i to their hosts together with the keys a_i^{pay}. If τ blocks were instead appended to the checkpoint block that do not contain the settlement transaction t, each player i's TEE will issue a refund by releasing the spending key a_i^{dep} to them.

See Appendix B for the formal description of the full protocol.

Parameters τ and τ'. Two different delay parameters are used in Auntie as opposed to the single τ_{final} used in FastKitten [16]. Indeed, FastKitten's timeout parameter τ_{final} corresponds directly to τ from Auntie. If the settlement transaction t was not confirmed by this time, players are eligible for a refund. Notably, both in Auntie and in FastKitten, there is a potential for race conditions: a player may apply for a refund (by calling $\mathsf{Player}(i).\mathsf{Refund}$ in Auntie or by publishing the penalty transaction tx_{p} in FastKitten) just as the operator broadcasts the settlement transaction. A malicious operator may even try to exploit this by waiting until τ blocks have been appended to C before broadcasting t. For this reason, we allow the players to call $\mathsf{Player}(i).\mathsf{Settle}$ even after they have called $\mathsf{Player}(i).\mathsf{Refund}$.

The settlement delay parameter τ' is there to satisfy the strong fairness property from Section 3.1. If player i could receive $(a_i^{\mathsf{pay}}, y_i)$ immediately (*i.e.*, if $\tau' = 0$), given t, then their colluding with the operator would be a concern. An adversary that has corrupted the operator and player i could decide whether or not they want to settle the contract based on the outcome of $\mathcal{F}$ for player i. Let δ be a network propagation delay (a parameter of the Zerocash ideal functionality, see [16]). If the time it takes for any coalition of parties to selfishly mine τ' blocks

on top of t is larger than the time it takes for the honest miners to mine $\tau + \delta$ blocks, then by the time malicious parties learn $(a_i^{\mathsf{pay}}, y_i)$, the contract will have been aborted due to timeout, making a_i^{pay} worthless and punishing the cheating operator who withheld t from the network. From now on, we assume $\tau' > \tau + \delta$.

Note that player i colluding with the operator may still learn their output y_i *and* get a refund if the operator withholds t. This is exactly identical to FastKitten, and the operator gets punished financially in both designs. Accounting for the economic value of the outputs y_i (*cf.* [18]) and, for example, the disutility player i gets from player j's learning y_j, is outside the scope of both [16] and the present work.

Fairness and Privacy. We now turn to proving Theorem 1 starting with the harder case of fairness. Auntie offers all the fairness guarantees of FastKitten plus strong fairness (recall Section 3.1) which guarantees parties must decide to be malicious ahead of time. In cryptographic terms, this corresponds to fairness under adaptive corruptions being immediately implied by fairness under static corruptions. In game-theoretic language, this makes an execution of Auntie a simultaneous-move game.

Lemma 1. *Assuming trusted execution environments realizing the TEE ideal functionality [16,32],* IND-CCA*-secure encryption* (Gen, Enc, Dec)*, and a blockchain realizing the Zerocash ideal functionality,* Auntie *is (strongly) fair.*

(Note that, in our hybrid model, we view Sign as an interface of the Zerocash ideal functionality, which in particular implies that signatures are unforgeable.)

Proof (sketch). Any aborts before Operator.Finalize is successfully called result in every honest party (player and operator alike) claiming and successfully proving timeout after τ blocks, and the outputs y_i, if even computed in the first place, remain escrowed with the TEEs and are never revealed to the players. Once Operator.Finalize has returned, the operator must act honestly and publish t lest they suffer a loss of Δ coins. Furthermore, if the operator does not publish t but sends it out of band to, say, player i, then player i can, having selfishly mined τ' blocks, convince their TEE to release y_i, but, by that time, a_i^{pay} will be worthless, as honest players will have long since claimed timeout.

Lemma 2. *Assuming trusted execution environments realizing the TEE ideal functionality [16, 32] and a blockchain realizing the Zerocash ideal functionality,* Auntie *is unobservable and unlinkable. Furthermore, if* Enc *is* IND-CPA*-secure, then* Auntie *is confidential.*

Proof (sketch). Intuitively, the ledger indistinguishability property of Zerocash [6] guarantees that contracts built on top of it are undetectable. Anonymity and confidentiality are achieved in Auntie by virtue of intermediating all deposits and payments by TEEs hosted by the players themselves and also escrowing inputs with them. When coupled with unlinkability of TEE remote attestation, the above hybrid makes the parties to a contract anonymous to each other and their repeated interactions unlinkable.

Together, Lemmata 1 and 2 imply Theorem 1. Observe also that Lemma 1 does not rely on any privacy properties of the underlying ledger except for the strong fairness case. The weaker BC ideal functionality of [16] could therefore be used instead of the Zerocash ideal functionality if one is only interested in fairness but not privacy or strong fairness. Finally, observe how the unobservability of Auntie additionally mitigates threats to liveness stemming from miner bribery and censorship.

4 TEEs of Today and Their Limitations

Trusted execution environment technology, however promising, is currently still very flawed (see, *e.g.*, [40]). This section surveys classes of vulnerabilities of TEEs and the scope of their impact on the properties of Auntie.

Side Channels. Side-channel leakage can largely be mitigated by relying on constant-time software with data-independent memory access patterns [22]. Such hardening is standard practice by now, at least for cryptographic algorithms such as signing or decryption. Notably, much of the secret data of the TEEs is only ever used once or twice, minimizing its exposure. *E.g.*, the output y_i, once computed, is only touched during encryption operator-side and then decryption player-side. Likewise, the spending key a_i^{dep}, once generated, is only used once to sign the settlement transaction, and the key a_i^{pay} is only used to derive A_i^{pay}. Relying on separate deposit addresses A_i^{dep} for each player and cooperatively signing the settlement transaction t (with each TEE signing its corresponding input) also ensures that player i exploiting any side channels can at best leak a_i^{dep} and "steal" their own coins back. The settlement transaction will then be invalid and the remaining players will be able to claim timeout. To also protect the operator in this case from losing their collateral of Δ, they may be given the serial numbers of the deposit outputs ahead of time to be able to tell if TEE guarantees were breached and the deposits redeemed ahead of time. Side channels may thus violate confidentiality (and strong fairness) of the contract when exploited by the operator but have no bearing on its (weak) fairness. This is notably not the case in FastKitten [16]. Counterintuitively, introducing more trusted hardware resulted in less reliance on its actually being trusted.

Clearly, unobservability of the contract and unlinkability of different sessions hold regardless of the data leaked during contract execution. Similarly, parties remain anonymous by virtue of the per-contract intermediate addresses A_i^{dep} and A_i^{pay} used.

Active Attacks. The operator's biasing the randomness used to evaluate $\mathcal{F}$ or produce $\hat{t}$ can be prevented by having the TEE rely on a pseudorandom number generator seeded with contributions from every player. Fault injection attacks (discounted altogether by, *e.g.*, [35] as "confined to a lab") may allow "skipping" branch statements and bypassing controls. Repeating computations and aborting

on any inconsistency may be used to mitigate these. Finally, full compromise of the TEE through extracting the attestation keys [13,39] enables parties to forge attestation reports and fool others into escrowing both coins and inputs x_i with an untrusted program merely simulating a TEE. As above, if player i's TEE is breached, then they may learn their output y_i unfairly but nevertheless cannot steal other players money, as it is attached to secret keys a_j^{dep} that player i's TEE has never seen. If the operator's TEE is breached instead, no guarantees about either privacy or fairness hold, same as in [16].

5 Conclusions and Future Work

The hybrid of trusted execution environments and Zerocash has not been explored before in the context of contract execution. As the Auntie protocol introduced in this work demonstrates, it offers an elegant solution that simultaneously addresses fairness, privacy, fungibility, as well as frontrunning and bribery prevention. It enables mutually distrustful parties' engaging in contracts without learning anything (non-negligible) about each other's identities while leaving no traces of the contract on the Zerocash blockchain and, in particular, ensuring the proceeds from the contract are indistinguishable from funds obtained through direct payments. The hybrid also lends itself to a game-theoretic treatment with privacy guarantees of Zerocash ensuring players gain no insight during contract execution, thus reducing many sequential-move games to simultaneous-move ones, where the players may as well decide on all choices *a priori*.[9]

When compared to Zexe [9] (whose goals are distinct but nonetheless overlap with those of present work), Auntie offers superior performance (comparable to plain Zerocash) and is, most importantly, immediately applicable to Zerocash as deployed today [21]. This is particularly relevant since replacing established designs may not be easy due to strong network effects of cryptocurrencies. When compared to FastKitten [16], Auntie provides superior privacy including full undetectability of the contract execution, anonymity of participants, unlinkability of their interactions, and confidentiality of deposits (on top of confidentiality of inputs x_i and outputs y_i already supported by FastKitten).

An interesting direction for future work is formalizing the conditions under which optimizations of the general framework are possible, *e.g.*, by relaxing the assumption that *every* player host a TEE, or that the operator make a deposit. For example, fair exchange (contingent payment) [18] can be achieved with only the buyer hosting a TEE that the seller escrows the commodity with and from which the seller also receives the settlement transaction t. It must, however, be the seller that receives t (notably, from the buyer's TEE), as only this allows the buyer's proving the contract has been settled to their TEE. There is clearly no single "operator" in this example, but readers are encouraged to convince themselves that the construction is nonetheless fair.

[9] This is assuming perfect envelopes for encryption and perfect zero-knowledge where applicable. Accounting for negligible leakage due to working under computational assumptions is outside the scope of this work.

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, : Secure multiparty computations on Bitcoin. Cryptology ePrint Archive **2013**, 784 (2013), https://eprint.iacr.org/2013/784
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, : Fair two-party computations via Bitcoin deposits. In: Proceedings of the 2014 Financial Cryptography and Data Security. pp. 105–121. Springer (2014), https://www.ifca.ai/fc14/bitcoin/papers/bitcoin14_submission_10.pdf
3. Austgen, J., Fabrega, A., Kelkar, M., Vilardell, D., Allen, S., Babel, K., Yu, J., Juels, A.: Liquefaction: Privately liquefying blockchain assets. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 1493–1511. IEEE Computer Society, Los Alamitos, CA, USA (May 2025). https://doi.org/10.1109/SP61157.2025.00156, https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00156
4. Avizheh, S., Haffey, P., Safavi-Naini, R.: Privacy-preserving FairSwap: Fairness and privacy interplay. Proceedings on Privacy Enhancing Technologies **2022**(1), 417–439 (2022). https://doi.org/10.2478/popets-2022-0021
5. Baldimtsi, F., Chalkias, K.K., Madathil, V., Roy, A.: SoK: Privacy-preserving transactions in blockchains. Cryptology ePrint Archive, Paper 2024/1959 (2024), https://eprint.iacr.org/2024/1959
6. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
7. Bentov, I., Ji, Y., Zhang, F., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-time cryptocurrency exchange using trusted hardware. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 1521–1538. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3363221, https://doi.org/10.1145/3319535.3363221
8. Bentov, I., Kumaresan, R.: How to use Bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 421–439. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
9. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 947–964 (2020). https://doi.org/10.1109/SP40000.2020.00050
10. Brickell, E., Li, J.: Enhanced Privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. IEEE Transactions on Dependable and Secure Computing **9**(3), 345–360 (2012). https://doi.org/10.1109/TDSC.2011.63
11. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) Financial Cryptography and Data Security. pp. 423–443. Springer International Publishing, Cham (2020)
12. Castejon-Molina, D., Vasilopoulos, D., Moreno-Sanchez, P.: MixBuy: Contingent payment in the presence of coin mixers. Proceedings on Privacy Enhancing Technologies **2025**(1), 671–706 (2025). https://doi.org/10.56553/popets-2025-0036
13. Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., Lai, T.H.: SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution. IEEE Security Privacy **18**(3), 28–37 (2020). https://doi.org/10.1109/MSEC.2019.2963021

14. Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., Song, D.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 185–200. IEEE (2019)
15. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 719–728. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134092, https://doi.org/10.1145/3133956.3134092
16. Das, P., Eckey, L., Frassetto, T., Gens, D., Hostáková, K., Jauernig, P., Faust, S., Sadeghi, A.R.: FastKitten: Practical smart contracts on Bitcoin. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 801–818. USENIX Association, Santa Clara, CA (Aug 2019), https://www.usenix.org/conference/usenixsecurity19/presentation/das
17. Dotan, M., Tochner, S., Zohar, A., Gilad, Y.: Twilight: A differentially private payment channel network. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 555–570. USENIX Association, Boston, MA (Aug 2022), https://www.usenix.org/conference/usenixsecurity22/presentation/dotan
18. Dziembowski, S., Eckey, L., Faust, S.: FairSwap: How to fairly exchange digital goods. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 967–984. CCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243857, https://doi.org/10.1145/3243734.3243857
19. Erwig, A., Faust, S., Riahi, S., Stöcker, T.: CommiTEE : An efficient and secure commit-chain protocol using tees. In: 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P). pp. 429–448 (2023). https://doi.org/10.1109/EuroSP57164.2023.00033
20. Frassetto, T., Jauernig, P., Koisser, D., Kretzler, D., Schlosser, B., Faust, S., Sadeghi, A.: POSE: Practical off-chain smart contract execution. In: 30th Annual Network and Distributed System Security Symposium (NDSS). The Internet Society (2023). https://doi.org/10.14722/ndss.2023.23118
21. Hopwood, D.E., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification (2026), https://zips.z.cash/protocol/protocol.pdf, accessed: 31 July 2026
22. Jean-Louis, N., Li, Y., Ji, Y., Malvai, H., Yurek, T., Bellemare, S., Miller, A.: SGXonerated: Finding (and partially fixing) privacy flaws in TEE-based smart contract platforms without breaking the TEE. Proceedings on Privacy Enhancing Technologies (PoPETs) **2024**(1), 617–634 (2024). https://doi.org/10.56553/popets-2024-0035, https://petsymposium.org/popets/2024/popets-2024-0035.php
23. Juels, A., Kosba, A., Shi, E.: The Ring of Gyges: Investigating the future of criminal smart contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 283–295. CCS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978362, https://doi.org/10.1145/2976749.2978362
24. Kaptchuk, G., Miers, I., Green, M.: Giving state to the stateless: Augmenting trustworthy computation with ledgers. In: Network and Distributed Systems Security (NDSS) Symposium. Internet Society (2019)
25. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE

Symposium on Security and Privacy (SP). pp. 839–858 (2016). https://doi.org/10.1109/SP.2016.55
26. Kumaresan, R., Moran, T., Bentov, I.: How to use Bitcoin to play decentralized poker. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. p. 195–206. CCS '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813712, https://doi.org/10.1145/2810103.2813712
27. Lee, J., Kim, S., Park, S., Moon, S.M.: RouTEE: Secure, scalable, and efficient off-chain payments using trusted execution environments. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC) (2024), https://www.acsac.org/2024/program/final/s161.html
28. Lind, J., Eyal, I., Pietzuch, P., Sirer, E.G.: Teechan: Payment channels using trusted execution environments (2017), https://arxiv.org/abs/1612.07766
29. Lind, J., Naor, O., Eyal, I., Kelbert, F., Sirer, E.G., Pietzuch, P.: Teechain: a secure payment network with asynchronous blockchain access. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. p. 63–79. SOSP '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3341301.3359627, https://doi.org/10.1145/3341301.3359627
30. Möser, M., Böhme, R., Breuker, D.: Towards risk scoring of Bitcoin transactions. In: Financial Cryptography and Data Security. pp. 16–32. Springer (2014)
31. Möser, M., Narayanan, A.: Effective cryptocurrency regulation through blacklisting (2019), https://maltemoeser.de/paper/blacklisting-regulation.pdf, manuscript
32. Pass, R., Shi, E., Tramèr, F.: Formal abstractions for attested execution secure processors. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 260–289. Springer International Publishing, Cham (2017)
33. Pfitzmann, A., Köhntopp, M.: Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology, pp. 1–9. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-44702-4_1
34. PwC: PwC global crypto regulation report 2025 (2025), https://legal.pwc.de/en/services/pwc-legals-eu-regulatory-compliance-operations/pwcs-global-crypto-regulation-report
35. Secret Foundation: Secret Network graypaper: The foundation of a confidential decentralized future (2020), https://scrt.network/graypaper, accessed: 4 November 2025
36. Thyagarajan, S.A., Malavolta, G., Moreno-Sanchez, P.: Universal atomic swaps: Secure exchange of coins across all blockchains. In: IEEE Symposium on Security and Privacy (SP). IEEE (2022)
37. Thyagarajan, S.A., Malavolta, G., Schmid, F., Schröder, D.: Verifiable timed linkable ring signatures for scalable payments for Monero. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) Computer Security – ESORICS 2022. pp. 467–486. Springer Nature Switzerland, Cham (2022)
38. Tramèr, F., Zhang, F., Lin, H., Hubaux, J.P., Juels, A., Shi, E.: Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 19–34 (2017). https://doi.org/10.1109/EuroSP.2017.28
39. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: extracting the keys to the intel SGX kingdom with transient out-of-order execution. In: Proceedings of the 27th USENIX Conference on Security Symposium. p. 991–1008. SEC'18, USENIX Association, USA (2018)

40. Van Schaik, S., Seto, A., Yurek, T., Batori, A., AlBassam, B., Genkin, D., Miller, A., Ronen, E., Yarom, Y., Garman, C.: SoK: SGX.Fail: How Stuff Gets eXposed. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4143–4162 (2024). https://doi.org/10.1109/SP54263.2024.00260
41. Wilde, A., Gruel, T.N., Soriente, C., Karame, G.: The forking way: When TEEs meet consensus. In: Network and Distributed System Security (NDSS) Symposium. Internet Society, San Diego, CA, USA (February 2025). https://doi.org/10.14722/ndss.2025.241934
42. Wüst, K., Matetic, S., Schneider, M., Miers, I., Kostiainen, K., Čapkun, S.: ZLiTE: Lightweight clients for shielded Zcash transactions using trusted execution. In: Goldberg, I., Moore, T. (eds.) Financial Cryptography and Data Security. pp. 179–198. Springer International Publishing, Cham (2019)

A Implementation and Evaluation

Auntie has been implemented using Zcash [21] and Intel SGX. The source code is available at `https://github.com/aCinal/auntie`. An example contract has been executed on the Zcash testnet. Its associated settlement transaction has ID

`121126653d34118dd71ef6e2e0461d29491e14b9b7a621497e798217891b5336`,

and was mined in block number 4235037. The payout keys a_1^{pay} and a_2^{pay} of the two players:

```
57a8a80f556ce7eb782b59e3296f8a716d264e6b4d48d63dc074bfa92692f831
9c8a88f7995c9950d9121b53e479558a09602f9aeec5e1970385e880bf81b519
```

can be used to inspect it.

B Full Protocol

The Auntie protocol is a tuple of $n + 1$ algorithms

$$(\mathsf{Player}_1, \ldots, \mathsf{Player}_n, \mathsf{Operator}),$$

each having access to the Zerocash ideal functionality ZC (with the same interface as the blockchain functionality BC in [16]) and the TEE ideal functionality TEE [32]. The ZC interface of interest are the post and getAll methods. $\mathsf{ZC.post}(t)$ publishes a transaction t to the blockchain after at most δ blocks, where δ is some parameter of the functionality. $\mathsf{ZC.getAll}(h)$ returns all blocks from the block height $h + 1$ to the tip of the blockchain.

To simplify exposition, let us introduce a number of notational shorthands having to do with TEEs. First, omit the installation call $\mathsf{TEE.install}$ that registers the Auntie program $P_{\mathsf{Auntie}}(n, \mathcal{F}, C, \Delta, \tau, \tau')$ with the TEE functionality. Similarly, omit the establishment of secure channels between the players' TEEs and that of the operator's. Note that, once these are in place, we no longer need the attestation quotes σ returned by $\mathsf{TEE.resume}$ (similarly as in [16]), so we may ignore these as well. Finally, since only a single TEE will be instantiated by each party, let us simply write $\mathsf{TEE}.f(x)$ instead of $\mathsf{TEE.resume}(E, f, x)$ for E the *enclave identifier* returned by $\mathsf{TEE.install}$ [32].

$\mathsf{Player}_i^{\mathsf{TEE},\mathsf{ZC}}(d_i, x_i, h_C)$

1 : $A_i^{\mathsf{dep}} \leftarrow \mathsf{TEE.Player}(i).\mathsf{Initialize}()$
2 : Make a transaction t_i sending d_i to A_i^{dep}
3 : $\mathsf{ZC.post}(t_i)$
4 : $c_i \leftarrow \mathsf{TEE.Player}(i).\mathsf{DepositAndInput}(t_i, x_i)$
5 : Send c_i to the operator
6 : Receive c_i' from the operator
7 : $(t_0, t_1, \ldots, t_n) \leftarrow \mathsf{TEE.Player}(i).\mathsf{GetDeposits}(c_i')$
8 : $B \leftarrow \mathsf{ZC.getAll}(h_C)$
9 : Verify $t_0, t_1, \ldots, t_n$ contained in B
10 : $c_i'' \leftarrow \mathsf{TEE.Player}(i).\mathsf{ConfirmDeposits}(1)$
11 : Send c_i'' to the operator
12 : $(a_i^{\mathsf{pay}}, y_i) \leftarrow \perp$
13 : **while** $(a_i^{\mathsf{pay}}, y_i) = \perp$ **do**
14 : $B' \leftarrow \mathsf{ZC.getAll}(h_C)$
15 : $(a_i^{\mathsf{pay}}, y_i) \leftarrow \mathsf{TEE.Player}(i).\mathsf{Settle}(B')$
16 : **endwhile**
17 : **return** $(a_i^{\mathsf{pay}}, y_i)$

Fig. 3: AUNTIE pseudocode for the player

$\mathsf{Operator}^{\mathsf{TEE},\mathsf{ZC}}(A_0^{\mathsf{pay}}, \Delta, h_C)$

1 : $A_0^{\mathsf{dep}} \leftarrow \mathsf{TEE.Operator.Initialize}()$
2 : Make a transaction t_0 sending Δ to A_0^{dep}
3 : $\mathsf{ZC.post}(t_0)$
4 : **foreach** $i \in \{1, \ldots, n\}$ **do**
5 : Receive c_i from player i
6 : **endforeach**
7 : $(t_1, \ldots, t_n, c'_1, \ldots, c'_n) \leftarrow \mathsf{TEE.Operator.ClearContract}(A_0^{\mathsf{pay}}, t_0, c_1, \ldots, c_n)$
8 : **foreach** $i \in \{1, \ldots, n\}$ **do**
9 : Send c'_i to player i
10 : Receive c''_i from player i
11 : **endforeach**
12 : $B \leftarrow \mathsf{ZC.getAll}(h_C)$
13 : Verify $t_1, \ldots, t_n$ contained in B
14 : $t \leftarrow \mathsf{TEE.Operator.Finalize}(c''_1, \ldots, c''_n)$
15 : $\mathsf{ZC.post}(t)$

Fig. 4: AUNTIE pseudocode for the operator