Absentia: Secure Multiparty Computation on Ethereum

Didem Demirag(✉) and Jeremy Clark

Concordia University, Montréal, Canada
d_demira@encs.concordia.ca

Abstract. This paper describes a blockchain-based approach for secure function evaluation (SFE) in the setting where multiple participants have private inputs (multiparty computation) that no other individual should learn. The emphasis of Absentia is reducing the participants' work to a bare minimum, where they can effectively have the computation performed in their absence and they can trust the result. While we use an SFE protocol (Mix and Match) that can operate perfectly well without a blockchain, the blockchain does add value in at least three important ways: (1) the SFE protocol requires a secure bulletin board and blockchains are the most widely deployed data structure with bulletin board properties (immutability and non-equivocation under reasonable assumptions); (2) blockchains provide a built-in mechanism to financially compensate participants for the work they perform; and (3) a publicly verifiable SFE protocol can be checked by the blockchain network itself, absolving the users of having to verify that the function was executed correctly. We benchmark Absentia on Ethereum. While it is too costly to be practical (a single gate costs thousands of dollars), it sets a research agenda for future improvements. We also alleviate the cost by composing it with Arbitrum, a layer 2 'roll-up' for Ethereum which reduces the costs by 94%.

1 Introduction

Consider the traditional setting for multiparty computation (MPC) with a twist: Alice and Bob each have some data, they would like to know the output from running an agreed-upon function on their data, each does not want the other (or anyone else) to learn their data, *and* they want to simply submit their data (*e.g.,* encrypted) to a trustworthy system and come back later for the result, which will always be correct. They are willing to pay for this service and they accept that, only in the worst case of full collusion between the operators of this service (called trustees), their inputs may be exposed—but a single honest trustee protects their privacy.

We assume the reader is familiar with blockchain technology, Ethereum, and smart contracts or decentralized apps (DApps). Can these technologies help? In theory? In practice? We seek to answer these questions through direct experimentation. The abstract above builds the argument for why blockchain can

M. Bernhard et al. (Eds.): FC 2021 Workshops, LNCS 12676, pp. 381–396, 2021.
https://doi.org/10.1007/978-3-662-63958-0_31

help: (1) it provides an integral point of coordination where trustees can post and track progress on the evaluation; (2) it provides an in-band solution for paying the trustees (in either a cryptocurrency like ETH or in a stablecoin pegged to the value of governmental currency like the USD) in a way that is contingent on their performance; and (3) the blockchain itself can serve as the public verifier and can reject any protocol proof that is not correct. When Alice and Bob retrieve the result (whether in plaintext or individually encrypted under their keys), they know it must be correct—otherwise it would not be there waiting for them.

Our experiments show that while in theory the idea is sound and we are able to successfully perform a secure function evaluation of a single logic gate (NAND gate) on Ethereum, the costs today are too prohibitive for it to be considered practical. We then turn to so-called layer-2 solutions and show that Arbitrum [14] can make Absentia substantially more practical (with room for further improvement).

1.1 Key Design Decisions

Note that we use the more precise term secure function evaluation (SFE) to describe the stateless, one-shot evaluation that Absentia provides. We think of SFE as a subset of secure multiparty computation (MPC)—a more general setting which includes stateful computations performed over time.

Design Decision: Trustee Model. In keeping with our priority for a submit-and-go protocol, someone has to perform the actual evaluation of the function on the inputs. We call these entities *trustees*. We require that the number of trustees (n) can be chosen independently of the number of inputs. In Absentia, we assume all trustees (n-out-of-n) participate (and can identify any that do not). However Absentia could be modified to allow the protocol to proceed if only a threshold (t out of n) of trustees participate—however, also reduces the number of trustees that need to collude to break the privacy of the protocol.

The remaining question is how can Alice and Bob find trustees they assume will not collude? We have several suggestions: (1) it could be based on personal connections; (2) perhaps commercial entities would emerge with either pre-established reputations or earn their reputation over time (similar to oracle providers today); confidence might increase if they offer legally enforceable terms of service; or (3) trustees could be picked at random from a large set of trustees. While (3) may not sound convincing, it is essentially same threat model as the anonymous web-browsing tool Tor which is trusted by many vulnerable users (perhaps Tor also uses flavours of (1) via its Entry Guard program).

Design Decision: Ethereum. While we are not the first to explore multi-party computation and its relationship to blockchain (see Sect. 2.1), we believe we are the first to implement an SFE/MPC protocol on a public, commonly used blockchain; namely, Ethereum. The first research question we ask is whether SFE/MPC is even feasible on Ethereum, given the heavy cryptography it uses.

Our paper establishes a benchmark that we hope to see improved through future research. Ethereum itself has scheduled scalability plans including Ethereum 2.0 (more transactions per second), and a lot of community resources are also being spent examining and implementing *layer 2* solutions that move blockchain functionality off of the main chain without sacrificing many of its security benefits. Technologies include *state channels*, *sidechains*, and *roll-ups* [12]. To experiment with these technologies, we also deploy and benchmark critical components of Absentia on Arbitrum [14], a recently proposed system for optimistic roll-ups (described more in Sect. 4). We now turn to another avenue for improvement, using state-of-the-art MPC protocols.

Design Decision: Mix and Match. Starting with Yao in 1982 [21], the question of how to securely evaluate a general function, when inputs are held by multiple people, has generated a rich body of literature in cryptography. In choosing an SFE/MPC protocol for the basis of Absentia, we looked for one with the following properties:

1. **Trustee model.** As justified above, we seek an SFE/MPC protocol that lets the input holders (*e.g.,* Alice and Bob) offload their inputs to a set of non-colluding trustees for evaluation.
2. **Publicly verifiable** (*a.k.a.* publicly auditable or universally verifiable). Many MPC/SFE protocols are in the semi-honest (*i.e.,* honest-but-curious) model. Some are resilient to covert or malicious adversaries. We require that not only can adversarial behaviour be detected by the participates in the protocol, but that it can be detected by anyone (*i.e.,* the public). This allows (a) Alice and Bob to offload the computational work to the trustees and still ensure the output is correct, even if they did not directly participate, and (b) Alice and Bob can go further and offload the verification itself to someone they trust—the Ethereum network in this case.
3. **Identifiable aborts.** If the protocol does not reach completion, anyone can establish which trustee aborted. Financial incentives can be attached to participation and timeliness.
4. **Elliptic curve operations.** While Ethereum can in theory implement different types of cryptography (RSA groups, integer-based discrete logarithms groups, lattices, *etc.*.), it has native support for its own cryptographic operations (ECDSA signatures) on the elliptic curve `secp256k1`. For ease of implementation, we prefer a SFE/MPC with the same cryptographic setting.
5. **Circuit type.** When the function to be evaluated is represented as a circuit, the circuit could be based on logic gates (*i.e.,* NAND gates) or arithmetic operations (*e.g.,* additions and multiplications in a modular group). We are indifferent to this design parameter.

One SFE protocol to meet our purposes is Mix and Match [13] and we chose it based on our familiarity with it. We are also aware that the state-of-the-art MPC protocols are based on a different paradigm—based on *Beaver triples* [4]—initiated by the SPDZ protocol [9,10] with many followups (HighGear is a recent

example [15]). While SPDZ uses lattice-based somewhat homomorphic encryption (SME), this is during a pre-computation phase and Absentia (for now) assumes all pre-computation has been validated. SPDZ also appears amenable to a trustee model and one paper explores a publicly verifiable variant [3], however since the authors do not compare themselves to Mix and Match, it would be a full research project to determine if it is indeed faster. We note that it is not obviously categorically faster—for example, by not requiring public key operations at all: the publicly verifiable variant uses Pedersen commitments extensively.

We are not aware of an explicit *proof* that Mix and Match is publicly verifiable, however every step of the protocol is covered by a trustee issuing a non-interactive zero knowledge proof and it is later assumed to be by the authors in their auction application [13]. Stated a different way, it appears that even when all trustees fully collude, trustees can only break privacy (and not integrity) with the exception of one sub-protocol, as noted by the authors [13], called the *plaintext equality test* (PET). Despite the caveat, many have used the PET protocol as if it is publicly verifiable (some making justifications based on statistical arguments). Recently it was shown these statistical arguments are not sufficient, but the PET protocol can be made verifiable, even when *all* trustees collude, with a simple additional check on the final output [17].

2 Preliminaries

2.1 Related Work

The blockchain literature has explored SFE and MPC in several regards. Perhaps the closest to Absentia is Enigma [22] which offers stateful MPC as a service. The original academic proposal utilizes a custom blockchain. Now as a commercial project, the emphasis is on providing generic smart contracts with privacy. Enigma runs on a Cosmos/Tendermint-based chain, with an Ethereum bridge contract that allows swapping crypto-assets. Absentia is different in the following regards: (1) users provide the circuit they want evaluated, (2) Absentia does not use trusted execution environments (TEE), and (3) we benchmark running natively on Ethereum. Like Enigma, Hawk also provides a privacy wrapper for contracts [16] based on succinct zero knowledge. A fair MPC is described as an application of Hawk but not implemented.

The literature has also explored moving computation off-chain while not losing privacy or correctness, however from the perspective of a single entity's secret data (*i.e.,* verifiable computing as opposed to SFE/MPC). Examples include Zexe [6], ZkVM [1], and Raziel [19]. Another research direction, initiated by Andrychowicz *et al.* [2], explores how blockchain technologies can support an off-chain MPC to provide fairness. By contrast, Absentia is performing the SFE on the blockchain. Closely related to SFE/MPC are zero knowledge proofs, whose uses in blockchain are now too prolific to adequately summarize here.

2.2 Background

We provide a basic overview of the Mix and Match protocol for secure function evaluation (SFE), while referring the reader to the original paper by Jakobsson and Juels for the full details [13]. Mix and Match uses a partially homomorphic encryption scheme; we instantiate it with additive exponential Elgamal [8]. We implement it over the elliptic curve `secp256k1` which is used natively by Ethereum (we describe later how this results in savings).

Mix and Match: Pre-computation. In a pre-computation stage, the following tasks are completed. First, a set of n trustees, identified by public keys, are chosen. A threshold of trustees needed to complete the protocol can also be chosen, however we implement the simplest case: 2-out-of-2 (we call this *distributed* as opposed to *threshold*). Next, the trustees use a distributed key generation (DKG) protocol for creating n shares of the decryption key, one for each trustee, as well as a single joint public key. Exponential Elgamal supports DKG and threshold decryption [18].

In Mix and Match, a circuit of the function to be evaluated is produced using multi-input and multi-output lookup tables. We evaluate a single binary NAND gate (a universal gate that can create any circuit) which corresponds to a lookup table with two binary inputs (one from Alice and one from Bob) and a single binary output. During a pre-computation stage, the circuit for the function is established as a sequence of lookup tables (the output from one table can be used as an input to another). Each element of each lookup table is individually encrypted under the trustees' public key (we denote an encryption of x as $[\![x]\!]$):

A	B	Out
$[\![0]\!]$	$[\![0]\!]$	$[\![1]\!]$
$[\![1]\!]$	$[\![0]\!]$	$[\![1]\!]$
$[\![0]\!]$	$[\![1]\!]$	$[\![1]\!]$
$[\![1]\!]$	$[\![1]\!]$	$[\![0]\!]$

The encrypted table is then permuted row-wise. Each trustee mixes the rows, rerandomizes each ciphertext, and proves in zero knowledge that the result is correct:

A	B	Out
$[\![0]\!]$	$[\![1]\!]$	$[\![1]\!]$
$[\![1]\!]$	$[\![0]\!]$	$[\![1]\!]$
$[\![1]\!]$	$[\![1]\!]$	$[\![0]\!]$
$[\![0]\!]$	$[\![0]\!]$	$[\![1]\!]$

Complete circuits of such tables can be pre-computed by the trustees before Alice and Bob provide their inputs. Practically speaking, if sets of trustees were pre-established, they could prepare circuits for commonly requested functions

and post them publicly. When Alice and Bob decide to do an SFE, they can choose the pre-computed circuit (produced by a specific set of trustees). For the purposes of this paper, we assume circuits have been pre-computed and verified. In the future we may extend Absentia to accept a circuit and complete set of proofs to verify its correct construction, but for this paper, we concentrate on building a verifier for the online phase.

Plaintext Equality Test (PET). Let $\langle [\![x]\!], [\![y]\!] \rangle$ denote two exponential Elgamal ciphertexts; encryptions of x and y respectively. The trustees will first compute $[\![z]\!] = [\![x - y]\!]$ using the additively homomorphic property. If the values are the same, $z = 0$; otherwise $z \neq 0$. Each trustee will choose a random $r_i \neq 0$, compute $[\![\hat{z}_i]\!] = [\![r_i * \hat{z}_{i-1}]\!]$ (where $\hat{z}_0 := z$) and prove correctness in zero knowledge. The resultant $[\![\hat{z}]\!] = [\![\prod r_i * z]\!]$ will still be $[\![0]\!]$ when $x = y$ and will encrypt a randomly distributed non-zero integer otherwise. (The original proposal [13] lets each trustee blind without using the result from the previous trustee—this adds asynchronicity but requires a critical security correction [17]). In the final step, the trustees decrypt and reveal $\hat{z}$. If $\hat{z} = 0$, the equality test returns `True`; and returns `False` otherwise.

Mix and Match: Online Phase. At this stage, Alice and Bob provide their inputs $\langle [\![a]\!], [\![b]\!] \rangle$. The trustees can begin with Alice's input $[\![a]\!]$ and they compute a PET between $[\![a]\!]$ and each ciphertext in the column corresponding to Alice's input. They do the same for Bob. They locate the row that returns true for every input column. The encrypted output(s) of this row can then be (1) transferred as an input to the next gate, (2) decrypted publicly if it is a final output, or (3) proxy re-encrypted for Alice (and/or Bob)—meaning it is obliviously and verifiably changed by the trustees from an encryption under the trustees' joint public key to an encryption under Alice's. For simplicity in Absentia, we implement (2). We illustrate for the previous example and $a = 1$ and $b = 0$:

A	B	Out
$\mathsf{PET}([\![a]\!], [\![0]\!]) = \mathtt{F}$	$\mathsf{PET}([\![b]\!], [\![1]\!]) = \mathtt{F}$	
$\mathsf{PET}([\![a]\!], [\![1]\!]) = \mathtt{T}$	$\mathsf{PET}([\![b]\!], [\![0]\!]) = \mathtt{T}$	$[\![1]\!]$ is selected
$\mathsf{PET}([\![a]\!], [\![1]\!]) = \mathtt{T}$	$\mathsf{PET}([\![b]\!], [\![1]\!]) = \mathtt{F}$	
$\mathsf{PET}([\![a]\!], [\![0]\!]) = \mathtt{F}$	$\mathsf{PET}([\![b]\!], [\![0]\!]) = \mathtt{T}$	

3 Absentia: System Design

High Level Flow. Figure 1 illustrates a high level overview of how participants interact with Absentia. The main contract of the system is the Absentia-DApp (`mixmatch.sol`), which can create sub-contracts: PET Sub-DApp (`PET.sol`). Note that Fig. 1 is stylized and the exact implementation might split/join certain function calls but it provides an accurate mental model of participation within the system.

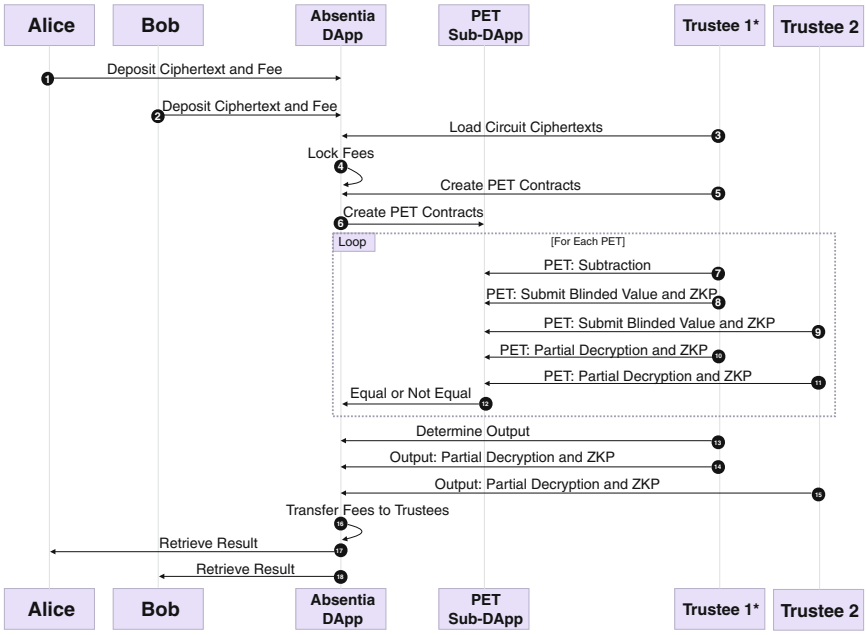

Fig. 1. Overview of Absentia.

At the beginning of the protocol, the contracts are deployed, identifying Alice, Bob, and the trustees (by Ethereum address). Alice and Bob both submit their encrypted input, and deposit fees that will be paid to the trustees for completing the protocol. We consider Absentia *submit-and-go* because Alice and Bob do not have to perform any other functions during the execution of the protocol.

Certain tasks are public operations that can be performed by anyone. For our analysis, we assume that Trustee 1 is the leader (denoted Trustee 1* with an asterisk) and always does these tasks. It is substantially more work, so it might improve the protocol to balance these operations between trustees or to compensate the leader more than the other trustees.

The actual Mix and Match operations done by each trustee is done off-chain using their share of the private key and other secrets (like randomizers) which are always offline. Ethereum is used to record the output of each step, record a zero-knowledge proof that the step was performed correctly, and to actually validate this proof. The DApp will reject any outputs accompanied by invalid or incomplete proofs. All proofs are Σ-protocols (specifically Schnorr [20] or Chaum-Pedersen [7]) made non-interactive with (strong [5]) Fiat-Shamir [11]. As this is not our main contribution, we refer the reader to the original paper by Jakobsson and Juels for the full details how these proofs are used in Mix and Match [13].

For each gate, the Absentia DApp creates enough instances of the PETs (*e.g.,* 8 instances for a binary gate) to perform the evaluation. The trustees then

interact with the PET contract, running each to completion (a state machine governs each step of the protocol). Note that for simplicity, Absentia requires the trustees to go in a specified order but the underlying protocol is amenable to some concurrency. Once enough PETs are complete that the output is determined, the leader can assert this to the Absentia DApp which will check the state of the PET contracts to confirm. The final output is staged for decryption by the trustees. Alice and Bob can find it on the Absentia DApp. For simplicity, the result is in plaintext however Absentia could be modified to support proxy re-encryption instead of decryption which would leave two final ciphertexts, encrypted respectively under public keys specified by Alice and Bob.

Payments. Absentia allows Alice and Bob to pay Trustee 1 and Trustee 2 upon completion of the protocol. We implement a simple proof-of-concept payment scheme while noting more elaborate schemes are possible. As implemented, Alice and Bob can deposit and withdraw ETH. The protocol cannot begin until their accounts hold enough to satisfy the fee (and if they hold more, the excess can be withdrawn at any time). Once the protocol begins, the funds for the fee are locked in escrow within the contract. If the protocol reaches finality, the funds are transferred to the accounts of Trustee 1 and 2 who can then withdraw (Note we use standard re-entrancy protection[1] on withdraws.) If the protocol times out without reaching finality, the fees are returned to Alice's and Bob's accounts.

An alternative incentive scheme might pay trustees gradually for each step of the protocol they complete and then a larger bonus for completing. Since Absentia can identify which trustee aborts the protocol (a useful feature that is not always possible in SFE/MPC protocols), trustees could also be required to post a payment (stake) to act as a fidelity bond. They financially commit to completing the protocol in a timely fashion and their stake is taken (slashed) if they do not.

Code Layout. Absentia is implemented in Solidity. All our code and tests are open source.[2] The trustees can perform their operations and generate their proofs in a language of their choice; we implement this in Mathematica (which we also use to generate test vectors for validating the Solidity code). `Mixmatch.sol` and `PET.sol` consists of 214 and 388 lines (SLOC) of Solidity code respectively. We adapt a standard library for elliptic curve operations.[3]

One optimization we implement concerns scalar multiplication over elliptic curves. Since Solidity is verifying proofs in Absentia, it only has to verify multiplications rather than perform them. Put another way, the trustee supplying the proof to Absentia already knows what the result of every multiplication is and can provide these values. As it turns out, it is cheaper to verify a multipli-

[1] Open Zeppelin's ReentrancyGuard.sol.
[2] https://github.com/MadibaGroup/2017-Absentia.
[3] Orbs' ECops.sol.

Table 1. Code size for mixmatch.sol

Code	Size (bytes)
Bytecode	27,178
Deployed	26,774
Initialisation and constructor code	404

cation than compute one by 'abusing' Ethereum's relatively inexpensive opcode for validating ECDSA signatures.[4]

Since Absentia generates a lot of PETs to perform the protocol, we implement this aspect with a factory design pattern. In this pattern, each PET is a stand-alone contract. The Mix and Match contract can create instances of these PET contracts and deploy them at new addresses. Our measurements (see below and Table 2) demonstrate that the factory pattern has certain drawbacks. `Mixmatch.sol` must deploy with a full copy of `PET.sol`'s bytecode in order for it to deploy instances of `PET.sol`. This results a contract size that is large. Also the function (Create Row) that creates (two) PETs each time it is called is the most expensive function in the system and costs $8,741,453$ gas (gas is Ethereum's metric for the cost of a computation).

All contracts enforce the order in which the functions can execute through state changes maintained within the contract. Key state changes emit events.

3.1 Measurements

Testing Platform. To test Absentia, we use Truffle on a local Ethereum blockchain. Our test files are included on the code repository. We also duplicated Absentia's functionality in Mathematica to help establish correctness.

Code size. The code size for `mixmatch.sol` is outlined in Table 1. When any Ethereum contract is first deployed, the constructor can only be run once. Thus the constructor code does not need to be referenced for further invocations and is not stored with the deployed bytecode (but can be found in the calldata of the deployment transaction).

When compiled, `mixmatch.sol` is $26,774$ bytes (plus a constructor of 404 bytes). Because of the factory design, this includes the bytecode to create `PET.sol` contracts. Ethereum limits contracts to ≈24KB (per EIP170).[5] We simply adjust Truffle's limit to allow us to benchmark it as a single contract. However it cannot be deployed on Ethereum today as is. Straightforward options to bring the code under the limit include: (1) taking `PET.sol` out of the contract and having the leader deploy each PET contract and load the addresses back

[4] V. Buterin, 2018. You can *kinda* abuse ECRECOVER to do ECMUL in secp256 k1 today.

[5] In 2016 when EIP170 was finalized, a 24KB contract could not deploy without crossing the block gas limit, however the gas limit has increased substantially since.

Table 2. Gas costs per function and who runs the function: Alice (A), Bob (B), Trustee 1 as the leader (T1*), or Trustee 2 (T2). Note that many functions are run more than once.

Contract	Function	Gas	Gas cost ($)
ec.sol	Deploy contract	595,517	31.94
Mixmatch.sol (Absentia DApp)	Deploy contract	6,091,398	326.75
	A&B: Load funds	28,040	1.50
	1*: Load outputs	300,798	16.13
	T1*: Create row	8,741,453	468.90
	T1*: Find matching row	37,547	2.01
	T1*: Find matching value	40,868	2.19
	T1*: Create final decryption	4,430,611	237.66
	A&B: Withdraw excess funds	41,110	2.21
	A&B: Withdraw funds	39,221	2.10
PET.sol (PET Sub-DApp)	Deploy Contract	4,681,858	251.14
	A&B or T1*: Load ciphertexts	304,668	16.34
	T1*: Subtraction	242,131	12.99
	T1*: Randomization ZKP	815,340	43.74
	T2: Randomization ZKP	393,561	21.11
	T1*: Partial dec ZKP	364,298	19.54
	T2: Partial cec ZKP	363,612	19.50
	T1*: Full cecryption	107,086	5.74
	T1*: Load final ciphertexts	173,945	9.33

Table 3. Cost for each participant.

	Alice	Bob	Trustee1*	Trustee2
Number of transactions	5	5	44	17
Total gas cost	1,246,712	1,246,712	52,952,603	6,420,996
Total cost in USD	66.87	66.87	2840.41	344.43

into `mixmatch.sol`; (2) move stateless functions to libraries; (3) split the contract up arbitrarily and use `delegatecall` to execute the pieces in a common context; or (4) find ways to optimize the code to reduce its size (it is academic, proof of concept code, and is very close to the limit, so this should be feasible).

Gas Costs. Table 2 provides the cost to deploy Absentia's two contracts and one library, as well as the gas costs of each function. Note that many functions are invoked more than once in a complete run of Absentia. The gas costs are as

Table 4. Cost of scaling absentia

Setting	Total gas
1 gate, 2 trustees	61,867,023
2 gate, 2 trustees	121,240,622
1 gate, 3 trustees	68,288,019

reported in Truffle's local network (Ganache). To convert gas into USD, we use 1 gas = 87 Gwei as recorded on Dec 01, 2020.[6] The price of ETH is $615.07 for the same date.[7]

As the leader of the protocol, Trustee 1 (T1*) has to perform more operations than the other participants. Table 3 shows the costs per participant. Particularly expensive tasks for the leader is loading all the ciphertexts for the circuit into the contract and initializing the memory needed, in particular for each PET, for the working memory. This is why, for example, Randomization ZKP is so expensive for T1 as compared to T2 (the code of both functions is identical but gas costs are 815, 340 versus 393, 561). Trustee 1 initializes many state variables (more expensive in Ethereum) that are not needed once the function completes; while trustee 2 overwrites the variables (less expensive in Ethereum). The next function, Partial Decryption, continues overwriting these variables.

Our design has some room for improvement. For example, in the current implementation, Alice and Bob have to deposit their inputs for each PET contract that is created (8 in total). A better design pattern (more consistent with Fig. 1) would have Alice and Bob deposit once in `mixandmatch.sol` and have the factory contract initialize the PETS with the correct values. Another improvement would aim to reduce the total transaction count for each participant by merging operations that are performed in a sequence by the same participant (we split them into logic blocks to better showcase what the gas was being spent on).

In Table 4, we show how Absentia scales with additional gates and additional trustees. If we want to evaluate a two gate circuit, Alice and Bob still perform the same number of transactions but nearly all of the rest of the functions are run twice as many times. Note that if the output of one gate is fed into the next gate, the leader (T1*) will load the inputs for the second gate. Going back to a single gate, increasing the number of trustees from 2 to 3 is not as expensive. Each additional trustee has a marginal cost equal to Trustee 2's cost in Table 3.

4 Absentia on Layer 2

4.1 Roll-Ups

A loose collection of technologies, called *Layer 2* solutions, have been proposed to address certain shortcomings of operating directly on Ethereum (*Layer 1*)

[6] Etherscan.
[7] Coinmarketcap.

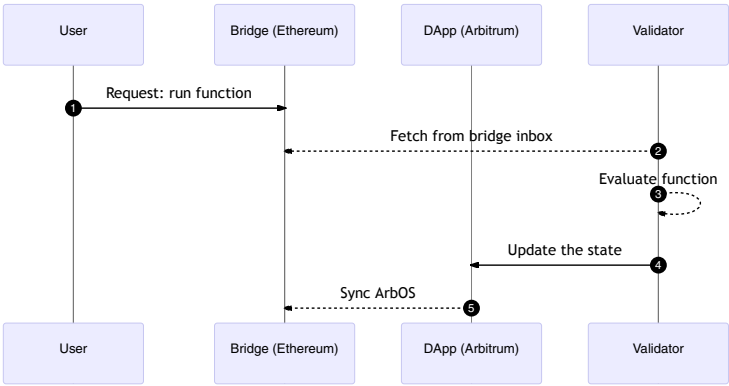

Fig. 2. Overview of arbitrum transaction submission.

or other blockchains [12]. These solutions generally strive for one or more of the following: reducing transaction latency, increasing transaction throughput, or reducing gas costs. In the case of Absentia, reducing gas costs is paramount. However Layer 2 solutions can also change the threat model; for Absentia, we require that Alice and Bob can trust the final output without having to verify any proofs themselves.

The most appropriate layer 2 technology for our requirements is called a *roll-up* which targets gas costs. In Ethereum, every transaction is executed (and thus validated) by every Ethereum node. In a roll-up, transactions are executed by off-chain nodes called *validators*. Validators try to convince the Ethereum network that the result of the transaction execution (*i.e.,* the state change of the EVM) is correct without the Ethereum nodes having to execute it.

Since Ethereum nodes cannot just ignore the Ethereum protocol's specifications for how to validate transactions, the roll-up cannot be implemented on Layer 1. Rather it is implemented inside its own DApp (Layer 2). This Layer 2 DApp is effectively a container, operating by its own custom consensus rules, for DApps that want roll-up functionality. The tradeoff is they are isolated from regular L1 DApps without some additional protocols (*e.g.,* interoperability support for currency/token transfers and external function calls). For Absentia, we do not require interoperability with L1 other than having a currency in L2 to pay the trustees.

There are at least two ways to convince on-chain participants that an off-chain computation was performed correctly. The first is to prove it with a succinct proof. SNARKs are one proof-type for general computations that are more efficient to verify than performing the computation itself. A second approach (called an *optimistic rollup*) is to have a validator assert the result and then allow for anyone to dispute it before finalizing it. Resolving disputes is always possible by having the Ethereum nodes perform the computation itself, but disputes can be settled in a more succinct way (see [14]). If Alice demonstrates that

Table 5. Comparison between deploying a plaintext equality test on Ethereum and deploying on arbitrum (via Ethereum). The links show the reader the actual transactions of a test-run on Kovan/Arbitrum's respective block explorers. Size is the calldata in bytes.

Function	Ethereum		Arbitrum				
	Tx	Gas	L1 Tx	L1 Gas	L2 Tx	L2 ArbGas	Size
Deploy ec	Link	1,103,372	Link	80,152	Link	1,304,481	4978
Deploy PET	Link	5,266,352	Link	386,079	Link	4,260,273	24,172
Load Ciphertexts	Link	305,309	Link	7869	Link	820,507	742
Subtraction	Link	260,729	Link	5469	Link	4,789,799	550
T1 Randomization ZKP	Link	819,877	Link	11,488	Link	10,972,720	644
T2 Randomization ZKP	Link	398,245	Link	11,440	Link	11,069,485	742
T1 Partial Dec ZKP	Link	366,636	Link	11,452	Link	10,692,786	742
T2 Partial Dec ZKP	Link	366,089	Link	11,512	Link	10,689,113	742
Full Decryption	Link	124,816	Link	6236	Link	4,258,675	422

a validator is wrong, the validator is financially punished and Alice is rewarded. Such validators do less work than Ethereum nodes (as well as validators that have to produce SNARKs)—therefore, optimistic rollups enable substantially lower gas costs.

4.2 Arbitrum

Arbitrum is a Layer 2 solution proposed in a *USENIX Security* paper [14] and now maintained as a commercial project by Offchain Labs. Currently, they operate an optimistic rollup on Ethereum. Instead of operating all Arbitrum contracts (called *ArbOS*) in a container DApp on Ethereum, ArbOS instead operates as a side-chain. A *bridge contract* on Ethereum serves as an interface between Ethereum and Arbitrum. Figure 2 shows how function calls work on Arbitrum. A user initiates a transaction on Ethereum to the Bridge Contract with the instruction to deploy a contract or run a function, along with all the data required for Arbitrum to perform this transaction. A validator sees new transactions in the inbox of the bridge, executes one and asserts the result to ArbOS. After a dispute period, the transaction is considered finalized. Periodically, the entire state of ArbOS is committed back to Ethereum. As all Arbitrum transactions are recorded on Ethereum, anyone can compute and compare the current ArbOS state.

4.3 Absentia on Arbitrum

Testing Platform. Arbitrum runs a testnet with a bridge on Ethereum's Kovan testnet. As mentioned above, Absentia is too large to deploy (as a factory contract) within Ethereum's contract size limit. To experiment with Arbitrum, we

implement only the PET sub-module as a standalone contract. We run the tests with Truffle. Instead of sending transactions to the Arbitrum bridge (as in Fig. 2), Arbitrum runs a service for developers where transactions are sent (off-chain) to a relay server (called an *Aggregator*) which will batch all pending transactions together as a single Kovan transaction to the bridge (and pay the gas). However we report the measurements as if the participants were sending the transactions themselves.

Gas Costs. Table 5 compares the cost of running a plaintext equality test (PET) on Ethereum (specifically Kovan testnet) and running it on Layer 2 (L2) with Arbitrum. Note the Ethereum numbers differ slightly from Table 2 as it is deployed on a different testnet (Kovan instead of private) and we modified it slightly to be a stand-alone DApp.

Arbitrum creates two transactions (recall Fig. 2): the Ethereum gas cost of relaying the (layer 1 or L1) transaction to the Arbitrum bridge, and the cost for the validator to execute the function, measured in ArbGas. The cost of the first Arbitrum transaction (*L1 Gas*) is paid with ETH but is invariant to its computational complexity. It is essentially only a function of its size (compare *L1 Gas* to *Size*). Note that the gas costs listed on the Kovan block explorer (links under *L1 Tx*) are for aggregated batches of transactions. We report what the cost would be to send it directly (not through an aggregator).

The ArbGas cost on Arbitrum should be similar to the gas cost on Ethereum, however validators do not run EVM bytecode directly. It is translated into Arbitrum virtual machine (AVM) bytecode which has its own opcodes and ArbGas costs. ArbGas has no market price currently. It is expected to be much cheaper than Ethereum's gas. In practice, the trustees could act as validators for Absentia transactions as they have to perform the computation anyways. Therefore we approximate arbgas as free.

A run of PET on Ethereum costs 9,011,425 gas (or 483.38 USD), while on Arbitrum the cost is 531,697 gas (or 28.52 USD). In this use case, Arbitrum reduces Ethereum gas costs by 94%.

5 Concluding Remarks

Ethereum can complement secure function evaluation protocols by enabling coordination, providing incentives, and enforcing correctness. Given recent developments in Ethereum toward performance and scalability, we felt it was an appropriate time to benchmark how expensive SFE is on Ethereum. Even though we expected it to be expensive, we did not imagine a single binary NAND gate would cost thousands of dollars on Ethereum. Most 'interesting' circuits are probably at least hundreds of gates, with many applications that would require many orders of magnitude more.

Despite this, we argue that Absentia is still an important research contribution. It proves the concept works, establishes a lower bound, and it sets a new research challenge: through improvements, how many gates can be evaluated

for, say, under $100 USD? Today it might be less than one but we are confident that future research can improve that number substantially. For example, our code can be further optimized; the latest MPC techniques can be applied; and Σ-protocols can be replaced with succinct zero-knowledge proofs. Meanwhile, Layer 1 and Layer 2 technologies will continue progressing.

Acknowledgements. We thank the reviewers who helped to improve our paper. J. Clark acknowledges support for this research project from the National Sciences and Engineering Research Council (NSERC)/ Raymond Chabot Grant Thornton/Catallaxy Industrial Research Chair in Blockchain Technologies and the AMF (Autorité des Marchés Financiers).

References

1. Andreev, O., Glickstein, B., Niu, V., Rinearson, T., Sur, D., Yun, C.: ZkVM: fast, private, flexible blockchain contracts. Technical report, Online (2019)
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: IEEE Symposium on Security and Privacy (2014)
3. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: SCN (2014)
4. Beaver, D.: Commodity-based cryptography. In: ACM STOC (1997)
5. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In: ASIACRYPT (2012)
6. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: Enabling decentralized private computation. In: IEEE Symposium on Security and Privacy (2020)
7. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO (1992)
8. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multiauthority election scheme. In: EUROCRYPT (1997)
9. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure mpc for dishonest majority-or: breaking the spdz limits. In: ESORICS (2013)
10. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO (2012)
11. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: CRYPTO, pp. 186–194 (1986)
12. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: Financial Cryptography (2020)
13. Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: ASIACRYPT (2000)
14. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: USENIX Security (2018)
15. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making spdz great again. In: EUROCRYPT (2018)
16. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: IEEE Symposium on Security and Privacy (2016)

17. McMurtry, E., Pereira, O., Teague, V.: When is a test not a proof? In: ESORICS (2020)
18. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: EUROCRYPT (1991)
19. Sánchez, D.C.: Raziel: Private and verifiable smart contracts on blockchains. Technical report, arXiv arXiv:1807.09484 (2018)
20. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptology **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725
21. Yao, A.C.: Protocols for secure computations. In: IEEE FOCS (1982)
22. Zyskind, G., Nathan, O., et al.: Decentralizing privacy: Using blockchain to protect personal data. In: IWPE (2015)