Raziel:
Private and Verifiable Smart Contracts on Blockchains

David Cerezo Sánchez*, David Evans†, Jonathan Katz‡
Calctopia*, University of Virginia†, University of Maryland‡
david@calctopia.com*, evans@virginia.edu†, jkatz@cs.umd.edu‡

(This paper is a first implementation of PCT/IB2015/055776)

September 9, 2017

Abstract

Raziel combines secure multi-party computation and proof-carrying code to provide privacy, correctness and verifiability guarantees for smart contracts on blockchains. Effectively solving DAO and Gyges attacks, this paper describes an implementation and presents examples to demonstrate its practical viability (e.g., private and verifiable crowdfundings and investment funds). Additionally, we show how to use Zero-Knowledge Proofs of Proofs (i.e., Proof-Carrying Code certificates) to prove the validity of smart contracts to third parties before their execution without revealing anything else. Finally, we show how miners could get rewarded for generating pre-processing data for secure multi-party computation.

*Corresponding author

1 Introduction

The growing demand for blockchain and smart contract[Sza97, BP17, CBB16, JE03, Hvi14, STM16] technologies sets the challenge of protecting them from intellectual property theft and other attacks[ABC16]: security, confidentiality and privacy are the key issues holding back their adoption[Mac16]. As shown in this paper, the solution must be inter-disciplinary (i.e., cryptography and formal verification techniques): code from smart contracts cannot be formally-verified before its execution with cryptographic techniques and conversely, formal verification techniques keep track of information flows to guarantee confidentiality but they don't provide privacy on their own[FLGR09, FPR11].

The availability of the proposed technical solution and its broad adoption is ultimately a common good: it's in the public interest to use smart contracts that respect the confidentiality and privacy of the processed data and that can be efficiently verified. Smart contracts have recently been heralded as "cryptocurrency's killer app"[Cas14], but to develop the next digital businesses based on the blockchain (e.g., the equivalents of Paypal, Visa, Western Union, NYSE) there is an urgent need for better protected smart contracts, a gap solved on this publication.

Contributions We propose a system for securely computing smart contracts guaranteeing their privacy, correctness and verifiability. Our main and novel contributions are:

- Practical formal verification of smart contracts: the proofs accompanying the smart contracts can be used to prove functional correctness of a computation as well as other properties including termination, security, pre-conditions and post-conditions, invariants and any other requirements for well-behaved code. A smart contract that has been fully formally verified protects against the subtle bugs such as the one that enabled millions to be stolen from the DAO[Dai16]. And when signed proofs conform to the specifications of a trusted organization, Raziel also prevents the execution of criminal smart contracts (i.e., Gyges attacks[JKS16]) solving this kind of attacks for the first time.

- A practical use of Non-Interactive Zero-Knowledge proofs of proofs/certifiable certificates is proposed (see section §5.2): the code producer can convince the executing party of the smart contract of the existence and validity of proofs about the code without revealing any actual information about the proofs themselves or the code of the smart contract. Although the concept of zero-knowledge proofs of proofs is not new[Blu87, BOGG$^+$90, Pop04], it can now be realised in practice for general properties of code.

- An outsourcing protocol for secure computation that allows offline parties and private parameter reuse is presented (see section §4.6): previous approaches don't allow all the parties to be offline[MGBF14, CMTB13, CLT14, CT16, BS14] or reusing encrypted values[KMR11, KMR12].

- We propose that miners should get rewarded for generating pre-processing data for secure multi-party computation (see section §4.7), in line with earlier attempts to replace wasteful PoWs[Kin13, MJS+14, BRSV17, ORP17].

Minimal set of functionalities We argue that the combined features here described are indeed the minimal set of functionalities that must be offered by a secure solution to protect computations on blockchains:

- A pervasive goal of blockchain technologies is the removal of trusted third parties: towards this end, the preferred solution in cryptography is secure multi-party computation, which covers much more than transactions. Moreover, public permissionless blockchains are executing smart contracts out in the open and any curious party is able to inspect the input parameters and detailed execution of any on-chain smart contract: again, MPC is the best cryptographic solution to protect the privacy of the computation of said smart contracts.
  - Since carrying out encrypted computations has a high cost, we introduce two improvements to reduce that cost: outsourcing for cloud-based blockchains (see section §4.6) and mining pre-processing data for secure multi-party computation (see section §4.7).
- Once encrypted smart contracts are being considered, a moral hazard arises: why should anyone execute a potentially-malicious encrypted smart contract from an anonymous party? Formal verification techniques such as proof-carrying code provide an answer to this dilemma: mathematical proofs about many desirable properties (termination, correctness, security, resource consumption, legal/regulatory, economic and functional, among others) can be offered before carrying out any encrypted execution.
  - To prevent that said proofs leak too much information about the smart contracts, a novel solution for zero-knowledge proofs of proofs is proposed (see section §5.2).

This paper describes an implementation discussing cryptographic and technical trade-offs.

2 Background

This section provides a brief introduction to the main technologies that underpin Raziel: blockchains, secure multi-party computation[Gol97] and formal verification.

Blockchains A blockchain is a distributed ledger that stores a growing list of unmodifiable records called blocks that are linked to previous blocks. Blockchains can be used to make online secure transactions, authenticated by the collaboration of the P2P nodes allowing participants to verify and audit transactions. Blockchains can be classified according to their openness. Open, permissionless networks don't have access controls (e.g., Bitcoin[Nak08] and Ethereum[But14a, Woo14]) and reach decentralized consensus through costly Proof-of-Work calculations over the most recently appended data by miners. Permissioned blockchains have identity systems to limit participation (e.g., Hyperledger Fabric[Cac16]) and do not depend on proofs-of-work. Blockchain-based smart contracts are computer programs executed by the nodes and implementing self-enforced contracts. They are usually executed by all or many nodes (*on-chain smart contracts*), thus their code must be designed to minimize execution costs. Lately, off-chain smart contracts frameworks are being developed that allow the execution of more complex computational processes.

Secure Multi-Party Computation Protocols for secure multi-party computation (MPC) enable multiple parties to jointly compute a function over inputs without disclosing said inputs (i.e., secure distributed computation). MPC protocols usually aim to at least satisfy the conditions of inputs privacy (i.e., the only information that can be inferred about private inputs is whatever can be inferred from the output of the function alone) and correctness (adversarial parties should not be able to force honest parties to output an incorrect result). Multiple security models are available: semi-honest, where corrupted parties are passive adversaries that do not deviate from the protocol; covert, where adversaries may deviate arbitrarily from the protocol specification in an attempt to cheat, but do not wish to be "caught" doing so ; and malicious security, where corrupted parties may arbitrarily deviate from the protocol. Multiple related cryptographic techniques (including secret sharing[Sha79], oblivious transfer[Rab05], garbled circuits, oblivious random access machines[Gol87]) and MPC protocols (e.g., Yao[Yao82, Yao86, LP04], GMW[GMW87], BMR[BMR90], BGW[BGW88] and others) have been developed since MPC was originally envisioned by Andrew Yao in the early 1980s.

Formal verification Formal verification uses formal methods of mathematics on software to prove or disprove its correctness with respect to certain formal specifications. Deductive verification is the preferred approach: smart contracts are annotated to generate proof obligations that are proved using theorem provers or satisfiability modulo theories solvers. In the software industry, formal verification is hardly used, but in hardware the high costs of recalling defective products explain their greater use: analogously, it's expected a higher use of formal verification methods applied to smart contracts compared to the general rate of use on the software industry given the losses in case of errors and exploitation[Dai16].

3 Model and Goals

Parties executing smart contracts need to protect their private financial information and obtain formal guarantees regarding their execution. The central goal for Raziel is to offer a programming framework to facilitate the development of formally-verified, privacy-preserving smart contracts with secure computation.

3.1 Threat Model and Assumptions

A conservative threat model assumes that parties wish to execute smart contracts but mutually distrust each another. Each party is potentially malicious and the smart contract is developed by one of the parties or externally developed. We assume that each party trusts its own environment and the blockchain; the rest of the system is untrusted. The threat model does not include side channel attacks or denial of service attacks.

3.2 Goals

A secure smart contract system should operate as follows: a restricted set of parties willing to execute a smart contract check the accompanying proofs/certificates to verify it before its execution. Then these parties send their private inputs to the nodes at the start of the execution of the smart contract; after that, the nodes run the smart contract carrying out the secure computation. Alternatively, the same parties may run the secure computation between themselves without the use of external executing nodes. Finally, the restricted set of parties obtain the results from the secure computation, which could be stored on the blockchain according to the rules of some consensus protocol.

4 Private Smart Contracts

Enabling smart contracts with secure computation techniques (e.g., secure multi-party computation, homomorphic encryption[RAD78], indistinguishability obfuscation[BV15]) is a key-step to the global adoption of blockchain technologies: encrypted transactions could be stored on the blockchain; secure computations could be carried out between distrustful parties; even the contract's code on the blockchain could be kept private.

Although it would be very profitable to sell/acquire smart contracts based on the value of their secret algorithms (see Markets for Smart Contracts) using homomorphic encryption (bootstrapping being a costly operation that can only be minimized[PV16, BLMZ16] and not avoided) and indistinguishability obfuscation (polynomial-time computable, but with constant factors $\geq 2^{100}$), are both considered currently infeasible due to concrete efficiency issues. In spite of impressive progress towards making these techniques practical[LMA+16, CMR17, CGGI17], they appear to be a long way from suitability for our purposes (i.e., *Obfustopia* is still a very expensive place, *Cryptomania* is much more affordable).

Another possible approach, which is being adopted by some blockchains and related technologies, is to rely on a trusted execution environment (most notably Intel's SGX[CD16]). Relying on trusted hardware is a risky bet and assumes a high level of trust in the hardware vendor. Several severe attacks have exposed vulnerabilities of SGX[BMD+17, SWG+17, MIE17, WKPK16, XCP15, SLKP17, LSG+16, BCLK17, Swa17, LJJ+17, Cor17]: all the current proposed and existing blockchains whose security rest on SGX aren't providing detailed explanations and proofs on how they are defending against these attacks. Unlike software bugs, new hardware would have to be deployed to fix these kinds of bugs, sometimes re-architecting the full solution.

Hence, our design employs secure multi-party computation. Secure multi-party computation is more mature than the fully homomorphic methods, and has a less trusting threat model than trusted execution approaches. MPC protocols have solid security proofs based on standard assumptions and efficient implementation. Current MPC technologies that can be used in production[ABPP15] include: secret sharing, garbled circuits, oblivious transfer and ORAMs. Secret sharing based schemes (i.e., many-round, dependent on the depth of the circuit) can be faster than garbled circuits/BMR (i.e., constant-round) in low-latency settings (LANs): when latency is large or unknown, it's better to use constant-round protocols[SZ13, BELO16].

A preferred approach is to use modular secure computation frameworks: for security reasons, at least two secure computation frameworks should be available, offering different cryptographic and security assumptions; in case the security of one of them is compromised, there will be a second option.

4.1 Off-chain computation

Even when using the fastest available secure multi-party computation techniques, the overhead would be very significant if secure computations would have to be executed on every full node of a blockchain, as have been previously proposed[But14b]. In Ethereum, at current prices (380 $/ETH, 21 Gwei/gas, 30/August/2017), multiplying or dividing 2 plaintext integers 10 million times costs:

$$5\frac{\text{Gas}}{\text{ops}} \cdot 10000000\,\text{ops} \cdot 0.000000001\frac{\text{ETH}}{\text{Gwei}} \cdot 21\frac{\text{Gwei}}{\text{Gas}} \cdot 380\frac{\$}{\text{ETH}} = \$399$$

and to store 32768 words of 256 bits (i.e. 1 megabit), it costs:

$$2000\frac{\text{Gas}}{\text{SSTORE}} \cdot 32768\,\text{ops} \cdot 0.000000001\frac{\text{ETH}}{\text{Gwei}} \cdot 21\frac{\text{Gwei}}{\text{gas}} \cdot 380\frac{\$}{\text{ETH}} = \$523$$

However, multiplying 2 plaintext integers 10 million times on a modern computer takes 0.02 seconds: since it costs $0.004/hour (Amazon EC2 t2.nano 1-year Reserved Instance), the same multiplications cost

$$\frac{\$0.004\,\text{hour}}{3600\,\text{seconds/hour}} \cdot 0.02\,\text{seconds} = \$0.000000022$$

It's order of magnitude more expensive, concretely

$$\frac{\$399}{\$0.000000022} = 18136363636.363636364$$

That is, more than 18 billions times more expensive: and as the price of Ether keeps rising, the costs of computation will also rise. On another note, current state-of-the-art secure computation executes 7 billion AND gates per second on a LAN setting [AFL$^+$16], but an 8-core processor executes 316 GIPS at the Dhrystone Integer Native benchmark (Intel Core i7-5960X), an approximate slowdown of

$$\frac{316\,\text{GIPS}}{7\,\text{Billion gates/second}} = 45.142857143\,\text{x}$$

which must not be additionally imposed to the overhead/overcost of a public permissionless distributed ledger: on the hand, it also means that there are overheads/overcosts being accepted which are higher than those imposed by secure computation or verifiable computation ($10^5 - 10^7$ for proving correctness of the execution [WB15]).

We address on-chain and off-chain secure computations separately:

- For on-chain secure computations, the validation of transactions happens through the replicated execution of the smart contract and given the fault assumption underlying the consensus algorithm. Because privacy-preserving smart contracts introduce a significant resource consumption overhead, it is important to prevent the execution of the same privacy-preserving smart contract on every node. We relax the full-replication requirement for PBFT consensus: a "query" command could be executed on a restricted set of nodes (large enough to provide consensus), and then an "invoke" store the result in the distributed ledger. Finally, achieving consensus is a completely deterministic procedure, with no possible way for differences.

- Regarding off-chain secure computations, simple oracle-like calls could be considered or other more complex protocols[ZH17, JT17, iEx17] for scalable off-chain computation.

4.1.1 Latency and its impact

As previously mentioned, MPC protocols can be roughly divided into two classes: constant-round protocols, ideal for high latency settings; and protocols with rounds dependent on the depth of the evaluated circuit, usually faster but only on very low latency settings. The present paper proposes the use of two protocol suites to get the maximum performance, independent of the latency:

- if all parties are on a low latency setting, they could use a very fast secret-shared protocol[AFL$^+$16, BLW08]

- but if parties are on a high latency setting, they must use a constant-round protocol[WRK17b, WMK16a]. Alternatively, they could outsource the secure computation to a cloud setting (see section §A), to use a faster secret-shared protocol.

4.2 Blockchain Solutions

The guarantees of privacy, correctness and verifiability are designed for the more threatening setting of public permissionless blockchains, although the smart contracts can also be used on private permissioned blockchains: it's also preferred that private blockchains keep their communications open to public blockchains to allow the use of smart contracts of public utility.

We considered two blockchains solutions:

- Hyperledger Fabric[Cac16] is a good fit for executing complex computational procedures like secure multi-party computation and formal verification techniques: including such complex procedures on its chaincodes (i.e. smart contracts) requires no special design considerations.

- Ethereum[But14a] is a public permissionless blockchain and integrating complex procedures on its smart contracts is much trickier: executed code consumes *gas*, thus the general pursuit to minimize the number of executed instructions. For complex procedures, oracles are the best option (external data providers, not to be confused with random oracles in cryptography or oracle machines in complexity theory). The initial purpose of oracles is to provide data that didn't belong in the blockchain (e.g., web pages) because decentralized applications that achieve consensus shouldn't rely on off-chain sources of information. But the mechanism can also be used for off-chain execution of complex code. The steps to incorporate an oracle are:

    1. Contract call to on-chain contract that executes complex computational procedures (e.g., secure multi-party computation). Said call must include enough *gas* and the correct parameters:
        (a) Parameters should be encrypted to prevent that other participants of the blockchain inspect them: a public key should be available for this purpose; only the executing oracle should be able to decrypt the parameters using the corresponding private key. Said executing oracle must be implemented as a trusted server of the calling party; otherwise, a more complex protocol involving outsourced oblivious transfer must be used.
        (b) Contributed *gas* is used to cover the costs of returning results to the calling contract.
    2. The executing oracle receives the contract call, decrypts the data and proceeds to the off-chain execution of the complex computational procedure.

3. The executing oracle returns back the results to the calling contract address.
4. Calling contract obtains the results: if large results are expected (e.g., some few kilobytes) it's better to store them on IPFS[Ben14] to prevent *gas* costs, and what would be returned is a pointer to said results.

Regarding Bitcoin[Nak08], although it would be possible to use oracles[Ori14] similar to the ones used for Ethereum, these are hardly used due to a combination of high transaction fees, high confirmation time and low transaction rate.

4.2.1 Alternative Protocols and Standards

Although this paper is focused on blockchains, it could be adopted to other financial standards such as:

- Financial Information eXchange[Com16]: messaging standard for trade communication in the equity markets, with presence in the foreign exchange, fixed income and derivatives market.
- Financial Products Markup Language (FpML)[SA17]: XML messaging standard for the Over-The-Counter derivatives industry.

4.3 Functionality and Protocol

The functionalities and protocols of this sub-section and section 6 constitute an open framework on which to instantiate different secure multi-party computation protocols, thus benefiting from upcoming research advances.

In this sub-section we present our secure protocol for private smart contracts, consisting of the following standard functionality:

Functionality 4.1: Secure computation of smart contracts

- **Parties:** $E_1, ..., E_N$, set of nodes N_i of a blockchain B
- **Inputs:** smart contract SC, private inputs from parties $E_1 : \overrightarrow{x}, E_2 : \overrightarrow{y}, ..., E_N : \overrightarrow{z}$
- **The functionality:**
  1. Secure computation of smart contract SC
  2. Results are returned and/or saved on the blockchain B
- **Output:** results from the secure computation $E_1 : \overrightarrow{r_1}, E_2 : \overrightarrow{r_2}, ..., E_N : \overrightarrow{r_N}$.

Functionality 4.1 (Secure computation of smart contracts) is implemented by the following protocol:

Protocol 4.2: Realising Functionality 4.1 (Secure computation of smart contracts)

- **Parties:** $E_1, ..., E_N$, set of nodes N_i of a blockchain B
- **Inputs:** smart contract SC, private inputs from parties $E_1 : \overrightarrow{x}, E_2 : \overrightarrow{y}, ..., E_N : \overrightarrow{z}$
- **The protocol:**
  1. Parties $E_1, ..., E_N$ proceed to execute the smart contract SC:
     (a) The private inputs $\overrightarrow{x}, \overrightarrow{y}, ..., \overrightarrow{z}$ and non-private inputs are sent to nodes N_i:
        i. Technically, this could be a HTTPS/REST call or an oracle call to executing nodes N_i.
        ii. The number of executing nodes N_i depends on the setting of the blockchain B (public/private permissionless/permissioned): that is, it could range from every node of the blockchain B to just a subset of nodes under PBFT consensus.
        iii. On permissioned blockchains, each executing node N_i should be a server owned/operated by the corresponding calling party E_i (i.e., the servers are assumed to be trusted and the adversary is on the network); but if the nodes are being run on a public cloud or any other server outside the full control of the corresponding calling party E_i, then a protocol for outsourcing secure computations must be used (see section §A).
     (b) The executing nodes N_i proceed to execute the smart contract SC: the execution command from parties $E_1, E_2, ..., E_N$ contains the preferred secure computation engine to be used (i.e., there could be multiple execution engines with different protocols under various security assumptions; if the parties don't agree, the secure computation will not be carried out).
  2. If consensus on the results of the previous computation is reached, then said results $\overrightarrow{r_1}, \overrightarrow{r_2}, ..., \overrightarrow{r_N}$ could be returned to executing parties $E_1, E_2, ..., E_N$ and/or written on the blockchain B.
- **Output:** results from the secure computation $E_1 : \overrightarrow{r_1}, E_2 : \overrightarrow{r_2}, ..., E_N : \overrightarrow{r_N}$.

The security of the protocol is proved on section §6.3.

4.4 Experimental Results

Table 1 summarizes the execution cost for several example applications, chosen for their economic significance:

1. Millionaire's Problem (i.e., determining who's got the bigger number without revealing anything else)

2. Second-price auction: sealed-bid auction not revealing the bids between the participants and without an auctioneer. The highest bidder wins, but the price paid is the second-highest bid[Vic61].

3. European Exchange Options: valuation of an option (the right, but not the obligation) using Margrabe's formula[Mar78] to exchange one risky asset for another risky asset at the time of maturity; this example is useful to hedge private portfolios of volatile crypto-tokens of physical assets. Suppose two risky assets with prices $S_1(t)$ and $S_2(t)$ at time t and each with a constant dividend yield q_i: we calculate the option to exchange asset 2 for asset 1 has a payoff $max(0, S_1(t) - S_2(t))$,

$$\begin{aligned} Price\,option &= S_1 e^{-q_1 t} N(d_1) - S_2 e^{-q_2 t} N(d_2) \\ \sigma &= \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2} \\ d_1 &= \frac{\ln\left(\frac{S_1}{S_2}\right) + \left(q_2 - q_1 + \frac{\sigma^2}{2}\right) t}{\sigma\sqrt{t}} \\ d_2 &= d_1 - \sigma\sqrt{t} \end{aligned}$$

   where ρ is the Pearson's correlation coefficient of the Brownian motion of S_i and σ_i are the volatilies of S_i. Private inputs are σ_i, S_i and q_i.

4. Currency Exchange Options: valuation of an option (the right, but not the obligation) using the model of Garman-Kohlhagen[GK83] to exchange one currency for another at a fixed price; this example is useful to hedge private portfolios of volatile crypto-currencies. Suppose two risky currencies with different interest rates but constant exchange rate: we calculate the calls and puts with the following equations,

$$\begin{aligned} Call &= S_0 e^{-\rho t} N(d_1) - X^{-rt} N(d_2) \\ Put &= X e^{-rt} N(-d_2) - S_0 e^{-\rho t} N(-d_1) \\ d_1 &= \frac{\ln\left(\frac{S_0}{X}\right) + \left(r - \rho + \frac{\sigma^2}{2}\right) t}{\sigma\sqrt{t}} \\ d_2 &= d_1 - \sigma\sqrt{t} \end{aligned}$$

   where r is the continuously compounded domestic interest rate, ρ is the continuously compounded foreign interest rate, S_0 is the spot rate, X is the strike price, t is the time to maturity and σ is the foreign exchange rate volatility. Private inputs are r, ρ and S_0 (with constant exchange rate).

5. Crowdfunding smart contract: a simple crowdfunding smart contract is considered, that checks if the minimum contribution target is reached and

then returns the raised amount, or 0 otherwise.

Algorithm 1 Crowdfunding smart contract

```
int crowdfund(int inputX, int inputY) {
int ret;
int sum = inputX + inputY;
int minimum = 1000;

  if ( sum >= minimum ) ret = sum; else ret = 0;

  return ret;
}
```

6. DAO-like Investment Fund: a simple emulation of an investment fund is considered, that check if the minimum contribution target is reached and then returns the principal compounded after a number of years.

Algorithm 2 DAO-like Investment Fund

```
float daoInvestFund(int inputX, int inputY) {
float ret;
int sum = inputX + inputY;
int minimum = 1000;

  if ( sum >= minimum ) ret = sum * (1 + (0.04/4))^(4*5); else ret = 0;

  return ret;
}
```

Example	AND Gates	Time A	Time B	Time C
Millionaire (int)	96	1	1	0.485
Second-price Auction (int)	192	1.1	1	0.862
European Exchange Options (float)	267507	810	1273.8	1185.49
Currency Call Options (float)	323529	979.6	1540.6	957.77
Crowdfunding smart contract(int)	128	1	1	0.458
DAO-like Investment Fund(int)	2144	6.5	10.2	0.458

Table 1: Execution times for application experiments. The times shown are the wall clock time in millisecond to complete the secure multi-party computation (i.e., from the initial oblivious transfers to the final reveling of the results) for various secure multi-party computation engines (A: semi-honest; B:malicious security; C: secret-sharing (Sharemind[BLW08] estimation)).

4.5 Modes of Interaction

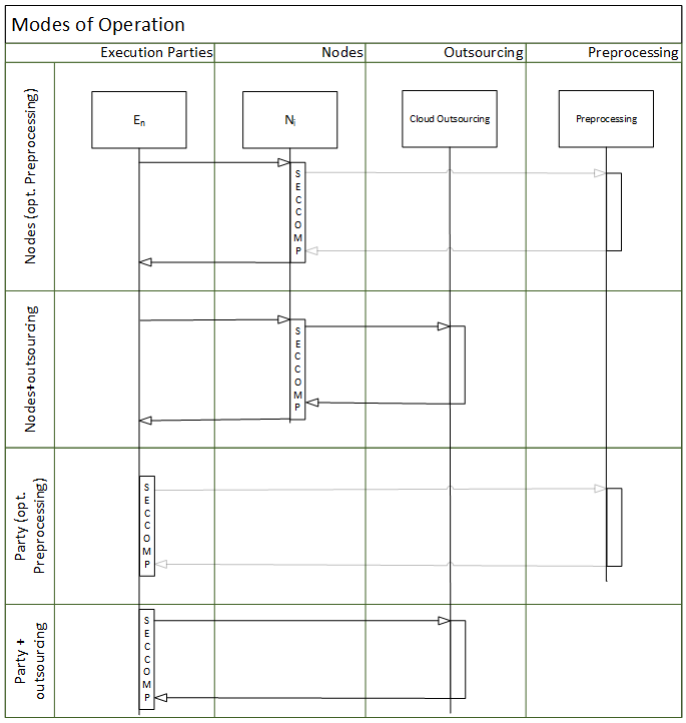

Figure 4.1: Supported modes of interaction

Secure computation can be carried on the nodes of the blockchain or on the parties themselves. Additionally, said secure computations can be outsourced to the cloud (see section §4.6) or use mined pre-processing data for secure multi-party computation (see section §4.7).

4.6 Outsourcing Secure Computations for Cloud-based Blockchains

The following scheme makes use of a multi-party non-interactive key exchange[BZ13, MZ17] to establish a shared secret between the computing parties and the executing nodes of the blockchain: replacing the NIKE protocol would require a PKI infrastructure between the computing parties and the executing nodes that will be used to establish shared secrets between them; it's viable but also more cumbersome within the context of public permissionless ledgers. A multiparty non-interactive key exchange (NIKE) scheme consists of the following algorithms:

- $Setup(M, N, \lambda)$: this algorithm outputs public parameters $params$, taking as input M, the maximum number of parties that can derive a shared key, N, the maximum number of parties in the scheme, and the security parameter λ.

- $Publish(params,\ i)$: this algorithm outputs the party's secret key sk_i, and a public key pk_i which the party publishes; inputs are public parameters $params$ and the party's index i.
- $KeyGen(params, i, sk_i, S, \{pk_j\}_{j \in S})$: this algorithm outputs a shared key k_S; inputs are public parameters $params$, the party's index i, the set $S \subseteq [N]$ of size at most M and the set of public keys $\{pk_j\}_{j \in S}$ of the parties in S.

As specified below, the functionality for outsourcing secure computations satisfies the following requirements:

- *offline parties*: no parties need to be involved when the outsourced secure computation is executed.
- *private parameters reuse*: parties don't need to re-upload their private parameters.
- *restricted collusion*: an adversary may corrupt a subset of the parties or the nodes of the blockchain, but not both[KMR11].

Functionality B.1: Outsourcing for Cloud-based Blockchains

- **Parties:** $E_1, E_2, ..., E_N$, set of nodes N_i of a blockchain B (N_G being the garbling node and N_E being the evaluator node)
- **Inputs:** smart contract SC, private inputs from parties $E_1 : \overrightarrow{x_i}, E_2 : \overrightarrow{x_i}, ..., E_N : \overrightarrow{x_i}$
- **The functionality:**
  1. $KeyExchange$: run a multi-party NIKE between parties $E_1, E_2, ..., E_N$ and nodes N_i and publish their public keys pk_i; from said public keys, a secret key k_S is derived.
  2. $SendPrivateParameters(E_i)$: send $(SendPrivateParameters,\ E_i)$ to N_E.
  3. $SECCOMP\,(SC)$: upon the reception of a computation request from party E_k for a function in smart contract SC, compute $y = SC\,(x_k, \{x_j | \forall j \in N : E_j\})$; send $(result, y)$ to E_k and $(SECCOMP, SC, E_k)$ to N_E.

Let Enc be a symmetric-key encryption algorithm secure against Chosen-Plaintext Attacks. The following protocol uses a pivot table during the secure computation, allowing the evaluator node to obliviously map the encoded inputs by the parties to the encoding expected by the circuit created by the garbling node. To implement Functionality B.1 (Outsourcing for Cloud-based Blockchains), the following protocol is proposed:

Protocol B.2: Realising Functionality B.1 (Outsourcing for Cloud-based Blockchains)

- **Parties:** $E_1, E_2, ..., E_N$, set of nodes N_i of a blockchain B (N_G being the garbling node and N_E being the evaluator node)
- **Inputs:** smart contract SC, private inputs from parties $E_1 : \overrightarrow{x_i}, E_2 : \overrightarrow{x_i}, ..., E_N : \overrightarrow{x_i}$
- **The protocol:**
  1. $KeyExchange$: run a multi-party NIKE between parties $E_1, E_2, ..., E_N$ and nodes N_i and publish their public keys pk_i; from said public keys, a secret key k_S is derived.
     (a) The setup phase is executed: $params := Setup(M, N, \lambda)$
     (b) Each party i runs $pk_i, sk_i := Publish(params,\ i)$ and publish pk_i
     (c) Each party i runs $k_S := KeyGen(params, i, sk_i, S, \{pk_j\}_{j \in S})$ to obtain the shared secret key
  2. $SendPrivateParameters(E_i : \overrightarrow{x_i}, k_S)$: party E_i chooses nonce n_i and for every bit of $\overrightarrow{x_i}$ computes $X_{il}^{\overrightarrow{x_i}[l]} = PRF_{k_S}(\overrightarrow{x_i}, l, n_i)$; then, party E_i sends these to N_E and also sends n_i. N_E stores $\left(\left(X_{i1}^{x_i[1]}, ..., X_{il}^{x_i[l]}\right), n_i\right)$.
  3. $SECCOMP\,(SC)$:
     (a) Secure computation at node N_G: the garbling node N_G compiles and garbles smart contract SC into the garbled circuit GC_{SC}. Then, for each party E_j and index l of the length of $\overrightarrow{x_i}$:
        i. Compute pivot keys: generate $s_{jl}^0 = PRF_{k_S}(0, l, n_j)$, $s_{jl}^1 = PRF_{k_S}(1, l, n_j)$.
        ii. Compute and save garbled inputs: using the pivot keys, encrypt $Enc_{s_{jl}^0}\left(w_{jl}^0\right)$ and $Enc_{s_{jl}^1}\left(w_{jl}^1\right)$; then save them into pivot table $P_q[j, l]$ in random order.
     (b) Secure computation at node N_E: for every bit of x_j and using the encoding $X_{jl}^{x_j[l]}$, decrypt the correct garbled values of each E_j from P_q; evaluate the garbled circuit GC_{SC} and send the output to N_G.
     (c) Result at N_G: decode the output to obtain the results $\overrightarrow{r_1}, \overrightarrow{r_2}, ..., \overrightarrow{r_N}$.
- **Output:** results from the secure computation $E_1 : \overrightarrow{r_1}, E_2 : \overrightarrow{r_2}, ..., E_N : \overrightarrow{r_N}$.

Theorem 1. *(Outsourcing for Cloud-Based Blockchains). Assuming secure channels between the parties $E_1, E_2, ..., E_N$ and the nodes N_i of the blockchain B, Protocol B.2 securely realises Functionality B.1 against static corruptions*

in the semi-honest security model with the garbling scheme satisfying privacy, obliviousness and correctness.

Proof. See Appendix A. □

4.7 Mining pre-processing data for Secure Multi-Party Computation

The Proof-of-Work of crypto-currencies consumes great amounts of computational power and electricity: 14TWh for Bitcoin[Dig17a] and 4.25TWh for Ethereum[Dig17b] just calculating hash functions. Miners could create pre-processing data for secure multi-party computation and be incentivised with crypto-tokens: 50-80% of total execution time is spent on pre-processing depending on the function/protocol, thus they would be profiting from "renting" their computational power to save significant amounts of computational time.

A recent paper considers the case of outsourcing MPC-Preprocessing[SSW17] to third parties and then computing parties reusing the pre-processed data (i.e., SPDZ-style authenticated shares) in the online phase: it's especially efficient if there is a subset of parties trusted by all the computing parties which can do all of the pre-processing and then distribute it to the computing parties. It's based on the re-sharing technique of [BGW88], but without using zero-knowledge proofs: it also fits into another recent protocol for secure multi-party computation[WRK17b, WMK16a] that is much more efficient for WAN networks than SPDZ derivatives, except that [WRK17b] uses BDOZ-style authenticated shares instead of SPDZ-style authenticated shares:

- BDOZ-style[BDOZ10] authenticated shares: for each secret bit x, each party holds a share of x; for each ordered pair of parties (P_i, P_j), P_i authenticates its own share to P_j. Specifically, when party P_i holds a bit x authenticated by P_j, this means that P_j is given a random key $K_j[x] \in \{0,1\}^k$ and P_i is given the MAC tag $M_j[x] := K_j[x] \oplus x\triangle_j$, where $\triangle_i \in \{0,1\}^k$ is a global MAC key held by each party. Let $[x]^i$ denote an authenticated bit where the value of x is known to P_i and is authenticated to all other parties: that is, $\left(x, \{M_k[x]\}_{k\neq i}\right)$ is given to P_i and $K_j[x]$ is given to P_j for $j \neq i$. An authenticated shared bit x is generated by XOR-sharing x and then distributing the authenticated bits $\left\{[x^i]^i\right\}$: let $\langle x\rangle := \left(x^i, \left\{M_j\left[x^i\right], K_i\left[x^j\right]\right\}_{j\neq i}\right)$ denote the collection of these authenticated shares for x.

- SPDZ-style[DPSZ11] authenticated shares: each party holds a share of a global MAC key; for a secret bit x, each party holds a share x and a share of the MAC on x. Specifically, a value $x \in \mathbb{F}_q$ is secret shared among parties P by sampling $(x_i)_{i\in P} \leftarrow \mathbb{F}_q^{|P|}$ subject to $x = \sum_{i\in P} x_i$ with a party i holding the value x_i; the MAC is obtained by sampling $(\gamma(x)_i)_{i\in P} \leftarrow \mathbb{F}_q^{|P|}$ subject to $\sum_{i\in P} \gamma(x)_i = \alpha \cdot x$ and party i holding the share $\gamma(x)_i$: let

$\langle x \rangle := \left((x_i)_{i \in P}, (\gamma(x)_i)_{i \in P} \right)$ to denote that x is an authenticated secret share value, where party $i \in P$ holds x_i and $\gamma(x)_i$, under a global MAC key $\alpha = \sum_{i \in P} \alpha_i$.

It's straightforward to adapt the protocol $\Pi_{Prep}^{R \to Q, \overline{A}}$ from [SSW17] to process BDOZ-style authenticated shares instead of SPDZ-style authenticated shares: thus, the authenticated shares generated by the offline pre-processing parties would be reshared amongst the computing parties as required before executing any secure multi-party computation and the pre-processing parties would be incentivised with crypto-tokens.

4.7.1 Security setting

Let E denote the set of n_E parties who are to run the online phase and O the set of n_O outsourcing parties that run the pre-preprocessing for the executing parties E (respectively Q and R in [SSW17]). Adversaries can corrupt a majority of parties in E and in O, but not all parties in E nor all parties in O: that is, each honest party in E believes that there is at least one honest party in O, but they may not know which one is honest. Let t_E denote the number of parties in E that are corrupt (resp. t_O in O): the associated ratios are denoted by $\epsilon_E = t_E/n_E$ and $\epsilon_O = t_O/n_O$. The executing parties E are divided into subsets $\{E_i\}_{i \in O}$ forming a cover, with a party in O associated with each subset: a cover is defined to be secure if at least one honest party in O is associated to one honest party in E.

In public permissionless blockchains, it's expected that there is no prior trust relation between parties in E and O: an efficient algorithm is offered in [SSW17] for assigning a cover to the network of parties so that the adversary can only win with negligible probability in the security parameter λ in the case where the covers are randomly assigned, and working out the associated probability of obtaining a secure cover. It assumes that each party in O sends to the same number of parties $l \geq \lceil n_E/n_O \rceil$ in E. The high-level idea of the algorithm is the following:

1. For each party in E, we assign a random party in O, until each party in O has $\lceil n_E/n_O \rceil$ parties in O assigned to it
2. For each party in O, we assign random parties in E until each party in O has l total parties which it sends to.

The probability to obtain a secure cover is given by

$$1 - \frac{t_E! \cdot (n_E - (n_O - t_O - 1)\lceil n_E/n_O \rceil)!}{n_E! \cdot (t_E - (n_O - t_O - 1)\lceil n_E/n_O \rceil)!} \cdot \left(\frac{\binom{t_E - \lceil n_E/n_O \rceil}{l - \lceil n_E/n_O \rceil}}{\binom{n_E - \lceil n_E/n_O \rceil}{l - \lceil n_E/n_O \rceil}} \right)^{n_O - t_O - 1}$$

In case where all but one party is corrupt in each E and O, then the probability to obtain a secure cover is given by l/n_E.

4.7.2 Preventing Sybil attacks

In the context of blockchains, Sybil attacks in which an attacker creates a large number of pseudonymous identities can easily be prevented: before running secure computations, any party could be required to deposit some arbitrarily high amount of money on a smart contract that would be confiscated in case any abnormal behaviour is detected (e.g., selective failure attacks, aborts, ...). Note that this simple technique maintains the anonymity of the parties and prevents that the computed pre-processing data gets intentionally wasted.

4.8 Other Applications

Applications of this technology can be found on many commercial/financial settings. Some of the most noteworthy are as follows:

- the most immediate application of secure multi-party computation is the removal of third parties and the financial industry has plenty of them: market makers, escrows, custodians, brokers, even banks themselves are intermediaries between savers and borrowers.
- crypto-banks: all the financial information contained within bank's databases could be encrypted with secure computation techniques. Users could make deposits, take loans and trade financial instruments without ever revealing their financial positions/transactions to inquiring third-parties.

Other applications mentioned in literature:

- economic/financial applications: financial exposures could be shared between mutually distrusting parties without revealing any confidential information to better control financial risks[EAA11]; credit scoring[DDN+15, Lie13]; facilitate the work of financial supervisors[FKOS13] without compromising confidentiality; protect the privacy of data in online marketplaces[CHK+11]; re-implement the stock market [Jut15] without a trusted auctioneer and no party learning the order book; remove escrows[Kum16] with claim-or-refund transactions and secure computation.
- game theory/mechanism design: remove trusted third-parties in tâtonnement algorithms[WC14] for one-time markets, allowing to privately share the utility function of the involved parties without revealing it and obtaining an incentive compatible protocol in the process; privacy-preserving auctions[AACM16, AV17]; more generally, implement mechanisms respecting privacy to obtain incentive-compatibility (e.g., auctions[NPS99, EL03]) or that incentivise data-driven collaboration among competing parties[AGP15].
- statistics: benchmark the performance of companies within the same sector[BTW11]; establish correlation and causation[BKK+15] between confidential datasets.

5 Verifiable Smart Contracts

After the DAO attack, there has been some work ([Hir17, BDLF+16, Fro16, PE16, HSR+17]) to include formal methods in the development of smart contracts: unfortunately, current solutions are very complex and cumbersome[Hir17, BDLF+16, PE16, HSR+17], almost equivalent to formally verifying assembly code. Only very high-level languages should be used to write smart contracts, not assembly-like ones (EVM): even the C language should be considered too low-level for these purposes because the required proofs must contain all kind of details about memory management and pointers.

Not all smart contracts need to be formally verified, and not even every part of their code should be. On permissioned blockchains, non-verified smart contracts will be much more accepted than on public permissionless blockchains.

Unlike other works that only consider the correctness of the computed output and resort to resource-consuming zero-knowledge proofs [BCG+14], the mathematical proofs considered here are multi-purpose: invariants, pre- and post-conditions, termination, correctness, security, resource consumption, legal/regulatory (e.g., self-enforcement), economic (e.g., fairness, double-entry consistency, equity), functional and any other desirable property that can be mathematically expressed.

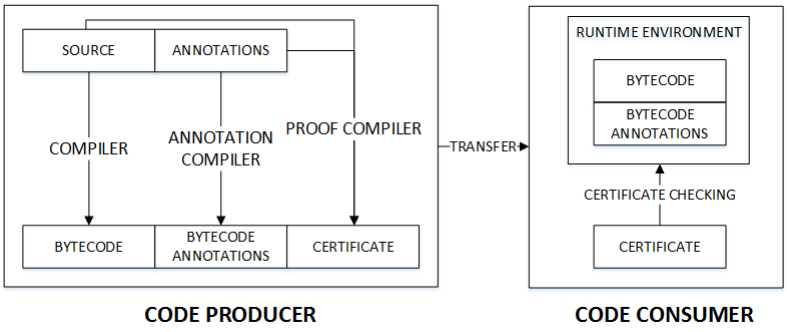

Figure 5.1: Proof-Carrying Code infrastructure

The solution of how to execute untrusted code from potentially malicious sources is not new, said problem was already considered for mobile code: Proof-Carrying Code[NL98] prescribes accompanying the untrusted code with proofs (certifiable certificates) that can be checked before execution to verify their validity and compare the conclusions of the proofs to the security policy of the code consumer to determine whether the untrusted code is safe to execute. On the positive side, no trust is required on the code producer and there is no runtime overhead during execution, but the code must be annotated (e.g., see pseudo-code annotation of the Program Listing below) and detailed proofs generated, a process which can be complex and costly. A novel variant combining PCC with Non-Interactive Zero-Knowledge proofs is proposed in the next sub-section 5.2.

Algorithm 3 Example pseudo-code with annotations

```
class Account {
  int balance; // invariant balance >= 0;
  // requires amt >= 0
  // ensures balance == amount
  Account(int amount) { balance = amount; }

  // ensures balance == acc.balance
  Account(Account _account) { balance = _account.balance(); }

  // requires amount > 0 && amount <= _account.balance()
  // ensures balance == \old(balance) + amount
  // && _account.balance == \old(_account.balance - amount);
  transfer(int amount, Account _account)
  { _account.withdraw(amount); deposit(amount); }

  // requires amount > 0 && amount <= balance
  // ensures balance == \old(balance) - amount
  void withdraw(int amount) { balance -= amount; }

  // requires amount > 0;
  // ensures balance == \old(balance) + amount
  void deposit(int amount) { balance += amount; }

  // ensures \result == balance
  int balance() { return balance; }
}
```

The verification of a program typically requires many annotations, at least a 1:1 ratio of lines of code against specifications: tools have been developed to automatically generate said annotations and are very useful in this setting. In case of using an interactive theorem prover, an extensive library of tactics must help to handle complex cases. Smart contracts must be written with the specific purpose of verification in mind, otherwise it becomes extremely complex to generate complete proofs[WSC+07]; regarding the size, effort and duration of the verification process, there is a strong linear relationship between effort and proof size[SJA+14] and a quadratic relationship between the size of the formal statement and the final size of its formal proof[MMA+15].

The most cost-effective option in some settings (e.g., permissioned blockchains) could be the publication of the annotated smart contracts to the blockchain and not using some of the more advanced options like the PCC toolchain or any kind of theorem prover, reversing the burden of proof to code consumers but in some sense helping them with the provided annotations. Therefore, there is a scale of Verification Levels when publishing smart contracts on blockchains:

1. Annotated smart contracts

2. Annotated smart contracts automatically tested using heuristics/concolic execution

3. Annotated smart contracts with full/partial proofs

4. Annotated smart contracts with certifiable certificates (Proof Carrying Code)

Ultimately, as an example of their applicability, the use of the proposed annotated smart contracts in combination with the zero-knowledge proofs of section §5.2 allows for an alternative way to implement the proofs of assets, liabilities and solvency of exchanges of [DBB+15].

5.1 Case Study

Algorithm 4 Crowdfund example processed with PCC toolchain

```
class Crowdfunding {
  int minimum = 1000;
  // requires 0 < n
  // ensures \result >= minimum
  public int crowdfund(int n, int[] inputs) {
    int sum = 0;
    // invariant 0 <= i && i <= n
    for (int i = 0; i < n; i++) {
      sum += inputs[i];
    }
    return sum;
  }
}
```

Proofs are written in Coq: the following execution statistics of the PCC toolchain are reported,

- Proof-generation time overhead: directly correlated to the size of the compiled program being analysed. It follows approximately the following formula on a modern laptop (Intel® Core™ i7-7500U 2.7Ghz):

$$time(bytecode\,size\,bs) = 1.5 + \left(\frac{bs}{1500}\right)\ secs$$

- Proof-verification time overhead: (less) correlated to the size of the compiled program being analysed. It follows approximately the following formula on a modern laptop (Intel® Core™ i7-7500U 2.7Ghz):

$$time(bytecode\,size\,bs) = 0.25 + \left(\frac{bs}{6000}\right)\ secs$$

- Certificate size overhead: 30%

	A	B	C
1	**Private And Verifiable Smart Contract**		
2	**European Exchange Option**		
3			
4	Time to Maturity	2	
5	Correlation Coefficient	0,4	
6			
7		**Asset 1**	
8	Price	100	
9	Volatility	0,2	
10	Dividend Yield	0,03	
11			
12		**Run A**	**Run B**
13	**Verification:**	✓	✓
14	**SECCOMP:**	**12,6679**	-PENDING-

Figure 5.2: Secure spreadsheet[Cal17] enabled for privacy-preserving computation displaying the result of private and verifiable smart contracts (e.g., cryptographically secure financial instruments and their derivatives)

Not much thought has been given to the rather practical question of what user interface should smart contracts use: calculations will be done on the returned values of smart contracts, even encrypted ones; and nested calculations on their inputs/outputs is also required. A spreadsheet enabled for secure computation fits all the given requirements, especially given that it's well accepted on the financial industry and many are trained on its use.

5.2 Zero-Knowledge Proofs of Proofs

Although it's possible to obfuscate certifiable certificates in such a way that de-obfuscation wouldn't be any easier[Dup08] while keeping the certifiable certificate sound and complete, this level of security isn't acceptable in a formal cryptographic model. When smart contracts are fully encrypted with homomorphic encryption/IO, certifiable certificates reveal too much information about the code and additional cryptographic protection is absolutely necessary: it's possible to generate zero-knowledge proofs of proofs[Blu87] (or certifiable certificates, which are shorter), in such a way that the code producer can convince the code consumer of the existence and validity of proofs about the code without revealing

any actual information about the proofs themselves or the code of the smart contract.

Classically, this would require to come by an interactive proof system[SGR85] were the code consumer is convinced, with overwhelming probability, of the existence and validity of proofs of the code through interaction with a code producer; then, a zero-knowledge proof system will be obtained using the methods of [BOGG$^+$90, IY88]: for a more detailed description on how to prove a theorem in zero-knowledge, see [Pop04]. Lately, advances in verifiable computation have produced advanced zero-knowledge proof systems to prove correctness of remote execution: although many of these results could theoretically be extended to prove general assertions about the code, the slowdowns for proving would be higher than the current $10^5 - 10^7$ for the very optimized case of proving correctness of executions[WB15].

Until the advent of methods to obtain zero-knowledge proofs from garbled circuits[JKO13] (i.e., general purpose ZK), zero-knowledge proofs have been difficult to come by. *ZKBoo*[GMO16], a later development of [JKO13], creates non-interactive zero-knowledge proofs for boolean circuits: this line of work is the preferred choice to obtain NIZK-proofs of the validity of certifiable certificates of smart contracts, because zk-SNARKs[BCCT12] would be very succinct, but with setup assumptions and much slower to proof (e.g., the circuit C_POUR from Zerocash[BCG$^+$14] has 4.109.330 gates and a reported execution time of 2 min). Lately, *ZKB++[CDG$^+$17]* provides proofs that are less than half the size than *ZKBoo* and *Ligero[HIV17]* four time shorter than *ZKB++*.

Using *ZKBoo*, the statement to be proved would be "I know a certifiable certificate such that $\phi(certificate) = true$" for a certificate validation circuit ϕ and L_ϕ the language $\{true | \exists certificate\, s.t.\, \phi(certificate) = true\}$ with soundness error 2^{-80}: minimizing the certificate validation circuit ϕ as much as possible would be the most important optimization.

Proof-Carrying Data A conceptually related technique to Proof-Carrying Code is Proof-Carrying Data[CT10]: in a distributed computation setting, PCD allows messages to be accompanied by proofs that said messages and the history leading to them follow a compliance predicate, in such a way that verifiers can be convinced that the compliance predicate held throughout the computation, even in the presence of malicious parties. PCC and PCD are complimentary since PCD can enforce properties expressed via PCC: interestingly, PCD could enable zero-knowledge privacy for PCC[CTV13], but at a greater efficiency cost.

6 Functionalities and Protocols

6.1 General Overview

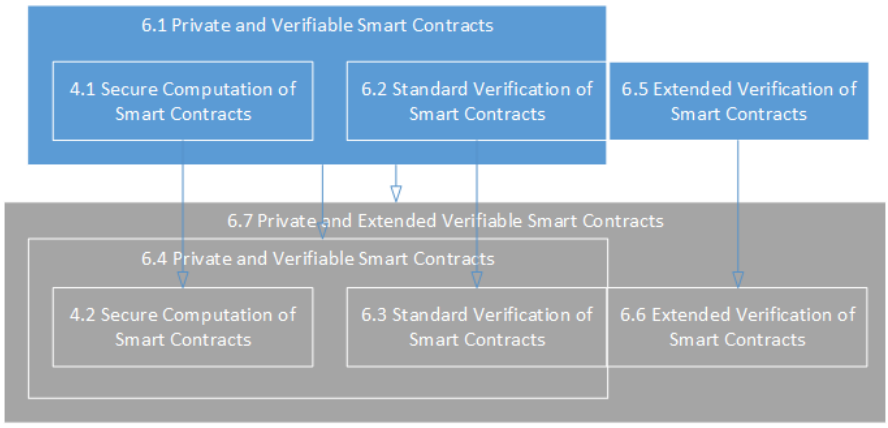

Figure 6.1: General overview of the functionalities (top) and realised protocols (bottom) described in this paper

6.2 Detailed Description

In this section we present our secure protocol for private and verifiable smart contracts, specified in the Functionality below:

Functionality 6.1: Private and Verifiable Smart Contracts

- **Parties:** $E_1, ..., E_N$, set of nodes N_i of a blockchain B
- **Inputs:** smart contract SC, private inputs from parties $E_1 : \overrightarrow{x}, E_2 : \overrightarrow{y}, ..., E_N : \overrightarrow{z}$
- **The functionality:**
  1. Parties $E_1, ..., E_N$ verify the smart contract SC
  2. Parties $E_1, ..., E_N$ securely execute the smart contract SC on nodes N_i
- **Outputs:** results from the secure computation $E_1 : \overrightarrow{r_1}, E_2 : \overrightarrow{r_2}, ..., E_N : \overrightarrow{r_N}$.

Our protocols for realising Functionality 6.1 consists of a standard verification functionality and a secure computation functionality (Functionality 4.1):

Functionality 6.2: Standard verification of smart contracts

- **Parties:** $E_1, ..., E_N$
- **Inputs:** smart contract SC
- **The functionality:**
  1. Parties $E_1, ..., E_N$ obtain the annotations/proofs/certificates of smart contract SC
  2. Verify the annotations, proofs and/or certificates available on the smart contract SC
  3. Check if the verified annotations, proofs and/or certificates are sufficient for the smart contract SC to be declared correct, according to the local policies of each party
- **Output:** *true* if the verification was correct, *false* otherwise.

To implement Functionality 6.2 (Standard verification of smart contracts), the following protocol is used:

Protocol 6.3: Realising Functionality 6.2 (Standard verification of smart contracts)

- **Parties:** $E_1, ..., E_N$
- **Inputs:** smart contract SC
- **The protocol:**
  1. Parties $E_1, ..., E_N$ download the annotations/proofs/certificates of smart contract SC and check their digital signatures.
     (a) If annotated smart contracts do not contain any kind of proofs or certificates, the annotations of the smart contract are automatically tested using an heuristic/concolic execution engine.
     (b) If annotated smart contracts contain full/partial proofs, they are locally regenerated and compared to the embedded ones.
     (c) If annotated smart contracts contain certifiable certificates, said certificates are checked.
  2. This step could be computationally expensive, so it only needs to be done the first time if the smart contract SC is not modified.
  3. Check if the verified annotations, proofs and/or certificates are sufficient for the smart contract SC to be declared correct, according to the local policies of each party
- **Output:** *true* if the verification was correct, *false* otherwise.

The previous protocols are combined in the following general execution protocol for private and verifiable smart contracts:

Protocol 6.4: Realising Functionality 6.1 (Private and Verifiable Smart Contracts)

- **Parties:** $E_1, E_2, ..., E_N$, set of nodes N_i of a blockchain B
- **Inputs:** smart contract SC, private inputs from parties $E_1 : \overrightarrow{x}, E_2 : \overrightarrow{y}, ..., E_N : \overrightarrow{z}$
- **The protocol:**
  1. The parties invoke Functionality 6.2 (Standard verification of smart contracts), implemented using Protocol 6.3, and obtain the verification status of the smart contract SC.
  2. The parties invoke Functionality 4.1 (Secure computation of smart contracts), implemented using Protocol 4.2 where party E_1 inputs $\overrightarrow{x}$, party E_2 inputs $\overrightarrow{y}$ and the rest of parties E_N their corresponding inputs. Parties receive results $E_1 : \overrightarrow{r_1}$, $E_2 : \overrightarrow{r_2}$ and the rest of parties their corresponding outputs $E_N : \overrightarrow{r_N}$.
- **Outputs:** results from the secure computation $E_1 : \overrightarrow{r_1}, E_2 : \overrightarrow{r_2}, ..., E_N : \overrightarrow{r_N}$.

6.3 Security Analysis

The security analysis follows the standard definition of static semi-honest security in the standalone setting[Gol04] and assumes semi-honest security of basic building blocks: Yao's protocol[LP04] and oblivious transfer accelerated with OT-extension[ALSZ13, IKNP03], implying the security for the Protocol 4.2 realising Functionality 4.1. Additionally, ORAMs[Gol87] could also be used to speed up dynamic memory accesses: also note that when the number of parties is huge, communication locality plays a central role and may be better achieved with ORAMs[BCP14, LO15]. The following theorem proves the general protocol 6.3:

Theorem 2. *(General Protocol). Protocol 6.4 realises Functionality 6.1 against static corruptions in the semi-honest security model augmented with verifiability of the code.*

Proof. Correctness and privacy of Protocol 4.2 (1.b) follows from Yao's protocol[LP04] and oblivious transfer accelerated with OT-extension[ALSZ13, IKNP03]; in case of a multi-party setting, correctness and privacy follows from one of the multi-party protocols (GMW[GMW87], BMR[BMR90], BGW[BGW88, AL11]); when using ORAMs, the security proof comes from the specific security proof of the chosen ORAM scheme (Circuit ORAM[WCS14]; Square-Root ORAM[ZWR$^+$16, GO96]); when using TLS, we make use of the composability of the security

proofs of said protocols[GMP+08, KMO+14, BFK+14]. Regarding verifiability, it follows from Protocol 6.3 realising Functionality 6.2. □

6.4 Extended verification of smart contracts

Trusted third parties (e.g., governments, central banks, regulating bodies) may provide specifications against which proofs must be generated. Thus, Gyges attacks[JKS16] can be prevented if parties and executing nodes require that all the executed smart contracts must adhere to said specifications and be digitally signed by trusted third parties: that is, prevent them from malicious hackers and others engaged in unlawful business practices while being compliant with certain regulations and laws (e.g., anti-money laundering laws). Comparing the next two diagrams provides a better understanding of the introduced differences:

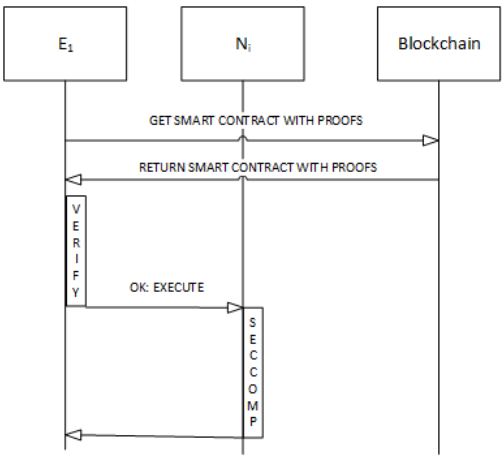

Figure 6.2: Standard verification of smart contracts

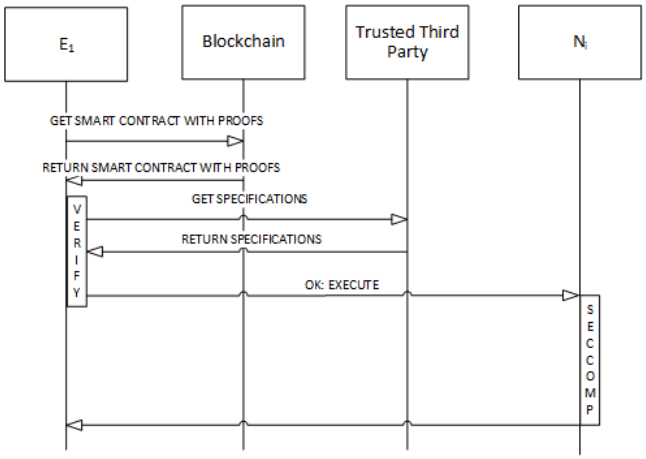

Figure 6.3: Extended verification of smart contracts

To formalize this concept, the following Functionality 6.5 is introduced:

Functionality 6.5: Extended verification of smart contracts

- **Parties:** $E_1, E_2, ..., E_N$, trusted third parties T_i
- **Inputs:** smart contract SC
- **The functionality:**
  1. Parties $E_1, E_2, ..., E_N$ obtain the annotations/proofs/certificates of smart contract SC
  2. Check digital signatures from trusted third parties T_i and verify conformance to their specifications: verify annotations, proofs and/or certificates available on the smart contract SC
  3. Check if the verified annotations, proofs and/or certificates are sufficient for the smart contract SC to be declared correct, according to the local policies of each party
- **Output:** *true* if the verification was correct, *false* otherwise.

The following Protocol 6.6 implements Functionality 6.5:

Protocol 6.6: Realising Functionality 6.5 (Extended verification of smart contracts)

- **Parties:** $E_1, E_2, ..., E_N$, trusted third parties T_i
- **Inputs:** smart contract SC
- **The protocol:**
  1. Parties $E_1, E_2, ..., E_N$ download the annotations/proofs/certificates of smart contract SC and check their digital signatures. Digital signatures from trusted third parties T_i are checked and conformance to specifications from said third parties T_i is tested: every party E_i has a security profile that specifies the mandatory/optional trusted signing parties and specifications that every smart contract must adhere to.
     (a) If annotated smart contracts do not contain any kind of proofs or certificates, the annotations of the smart contract are automatically tested using an heuristic/concolic execution engine. At least, it should be digitally signed by a trusted third party.
     (b) If annotated smart contracts contain full/partial proofs, they are locally regenerated and compared to the embedded ones. Some of said proofs must conform to specifications from third parties.
     (c) If annotated smart contracts contain certifiable certificates, said certificates are checked. Some of said certifiable certificates must conform to specifications from third parties.
  2. This step could be computationally expensive, so it only needs to be done the first time if the smart contract SC is not modified.
  3. Check if the verified annotations, proofs and/or certificates are sufficient for the smart contract SC to be declared correct, according to the local policies of each party.
- **Output:** *true* if the verification was correct, *false* otherwise.

Finally, the general Protocol 6.4 is reviewed with the new Protocol 6.6 for the extended verification of smart contracts:

Protocol 6.7: Realising Functionality 6.1 (Private and Extended Verifiable Smart Contracts)

- **Parties:** $E_1, E_2, ..., E_N$, set of nodes N_i of a blockchain B, trusted third parties T_i
- **Inputs:** smart contract SC, private inputs from parties $E_1 : \overrightarrow{x}, E_2 : \overrightarrow{y}, ..., E_N : \overrightarrow{z}$
- **The protocol:**
  1. The parties invoke Functionality 6.5 (Extended verification of smart contracts), implemented using Protocol 6.6, and obtain the extended verification status of the smart contract SC.
  2. The parties invoke Functionality 4.1 (Secure computation of smart contracts), implemented using Protocol 4.2 where party E_1 inputs $\overrightarrow{x}$, party E_2 inputs $\overrightarrow{y}$ and the rest of parties E_N their corresponding inputs. Parties receive results $E_1 : \overrightarrow{r_1}, E_2 : \overrightarrow{r_2}, ..., E_N : \overrightarrow{r_N}$.
- **Outputs:** results from the secure computation $E_1 : \overrightarrow{r_1}, E_2 : \overrightarrow{r_2}, ..., E_N : \overrightarrow{r_N}$.

The following theorem finally proves the extended general protocol 6.7:

Theorem 3. *(Extended General Protocol). Protocol 6.7 realises Functionality 6.1 against static corruptions in the semi-honest security model augmented with extended verifiability of the code.*

Proof. Correctness and privacy of Protocol 4.2 (1.b) follows from Yao's protocol[LP04] and oblivious transfer accelerated with OT-extension[ALSZ13, IKNP03]; in case of a multi-party setting, correctness and privacy follows from one of the multi-party protocols (GMW[GMW87], BMR[BMR90], BGW[BGW88, AL11]); when using ORAMs, the security proof comes from the specific security proof of the chosen ORAM scheme (Circuit ORAM[WCS14]; Square-Root ORAM[ZWR+16, GO96]); when using TLS, we make use of the composability of the security proofs of said protocols[GMP+08, KMO+14, BFK+14]. Regarding verifiability against Gyges attacks[JKS16], it follows from Protocol 6.6 realising Functionality 6.5. □

6.5 Malicious security

Smart contracts are compiled to efficient Boolean circuits that could be executed on multiple secure computation frameworks with the added benefit of obtaining malicious security[WMK16b, WRK17a, WRK17b, WMK16a]: protocols and theorems above can trivially be updated with their security definitions and proofs. In case of using ORAMs, security against malicious adversaries comes from specialized versions of said protocols [AHMR14, Mia16, HY16].

6.6 Private Function Evaluation

In this setting, the smart contract itself is kept secret: that is, only one of the parties knows the function $f(x)$ computed by the smart contract, whereas other parties provide the input to the private function without learning about f besides the size of the circuit defining the function and the number of inputs and outputs. Private function evaluation can be implemented using secure function evaluation[AF90, KS08] by securely evaluating a Universal Circuit[Val76] that is programmed by the party knowing the function $f(x)$ to evaluate it on the other parties' inputs: the security follows from that of the secure function evaluation protocol that is used to evaluate the Universal Circuit.

Recently, Valiant's Universal Circuit has been implemented and proven practical[KS16, GKS17]: its optimal size was proven to be $\Omega(k \log k)$[Val76] and its recent implementation further improved it by (at least) $2k$. Although it's a considerable blowup in size, late MPC protocols provide very efficient pre-processing phases that could ameliorate execution times: online times are only 1-2% of the total execution time in the WAN setting[WRK17b, WMK16a], and 2-5% in the LAN setting. Since the generated universal circuit description UC_{u,v,k^*} is public to all parties[KS16, GKS17], both function-dependent and function-independent pre-processing[WRK17b, WMK16a] could be used in the setting of private function evaluation for maximum speedup.

7 Economic Impact and Legal Analysis

Previous legal analysis[Dam16] has considered how secure computation fits current legal frameworks, although it's much more interesting to evaluate how it could alter legal frameworks and its economic impact.

7.1 Data Privacy as Quasi-Property

Quasi-property interests[Alg12] refer to situations in which the law seeks to simulate the functioning of property's exclusionary apparatus, through a relational entitlement mechanism, by focusing on the nature and circumstances of the interaction in question, which is thought to merit a highly circumscribed form of exclusion. The term first appeared in a SCOTUS decision in International News Service v. Associated Press[SCO18], in which Justice Pitney recognized the right of an information gatherer to prevent a competitor from free riding on the original gatherer's labor for a limited period of time: more specifically, it's limited in that it would only ever exist between the two parties in question and never in the abstract against the world at large.

Quasi-property allows to effectively treat data privacy as a property right[Sch12] and it's the closest analogue in American law that grants individuals a property right in their personal data, as privacy has always been considered in the privacy-preserving literature in cryptography: in fact, the quasi-property view of privacy-preserving computation has stronger grounding than European database rights[RS97]. This approach provides a common framework that underlies all of

the privacy torts, while avoiding the need to define privacy in such a way that it describes every injury that the law recognizes as an invasion of privacy and that can be generally categorized in four harms: (1) information collection; (2) information processing; (3) information dissemination, and (4) invasion.

7.2 A Solution to Arrow's Paradox

Arrow's Paradox[Arr62] states that "there is a fundamental paradox in the determination of demand for information; its value for the purchaser is not known until he knows the information, but then he has in effect acquired it without cost". *Ex-ante*, the purchaser cannot value the original information since it can only be known after it has been revealed; *ex-post*, the purchaser could not compensate the seller and disseminate it for free. Due to the inherent properties of information (non-excludable, non-rivalrous), markets for information cannot exist in the absence of intellectual property rights[GS03] because the original producer/inventor of any information loses the monopoly on it after the information is revealed; regarding financial information, the finance literature generally agrees in that the troubles in informational trading explain financial intermediation[LP77, RT84, All90, AP90].

Secure computation techniques offer a practical solution to Arrow's Paradox: distrustful third-parties can compute on private information without disclosing it, allowing its valuation while preventing intellectual property theft; this is significantly better than previously known methods based on partial the revelation of information[AY94, AY02, AY05]. More generally, secure computation techniques make information excludable while maintaining its non-rivalrousness: that is, the original producer/inventor of the information retains market power over it while maintaining its costless resale (i.e., the marginal cost of an additional digital copy is zero), theoretically allowing for digital goods of infinite value that bypass Coase conjecture[Coa72] since it's also possible to prevent that first time purchasers resell the information they have queried/acquired[Mut90, NQ91], although in practice having to account for the costs of the slowdown introduced by secure computation techniques.

7.3 Expansion of Trade Secrecy

While software copyright and patentability protection have been weakened, trade secrecy stands firm. As defined in the United States' Uniform Trade Secrets Act[Com85]: "Trade secret means information, including a formula, pattern, compilation, program, device, method, technique, or process, that: (i) derives independent economic value, actual or potential, from not being generally known to, and not being readily ascertainable by proper means by, other persons who can obtain economic value from its disclosure or use, and (ii) is the subject of efforts that are reasonable under the circumstances to maintain its secrecy". The definition on the European Union's Trade Secrets Directive[Com85] implicitly includes computer programs, financial innovations, lists of customers, business statistics and many other types of secret information.

There is no definitional uncertainty in trade secret law: unlike copyright, there is no exclusion for functionality; unlike patents, there is no exclusion for abstract ideas[Ris16]. There isn't any uncertainty about the exact contours of their extension either, such as in the debate of the mutual exclusivity of copyright and patents protections[UU91]. In some cases, widely distributed software may remain a trade secret if the license agreement requires confidentiality and return upon non-use (see Data Gen. Corp. v. Grumman Systems Support Corp.[USCoA94]).

Although software might be reverse engineered (i.e., an acceptable way to discover a trade secret that prevents its general use for protecting software), secure computation techniques effectively hinder and/or completely forbid reverse engineering. Trade secrets can be justified as a form, not of traditional property, but of intellectual property in which secrecy is central[Lem08] and may serve the purposes of IP law better than more traditional IP rights.

7.4 Verifiability and Self-Enforcement

The term *Lex Cryptographia*[AW15] designates a new body/subset of law, containing rules administered through smart contracts and decentralized autonomous organizations: this concept is inspired on *Lex Mercatoria*, an old subset of customs that became recognized as a customary body of law for international commerce. Smart contracts are just software programs with very specialized functionalities intended to replace paper contracts: the practice of law could thus follow the path of software, with smart contract programming languages becoming more powerful and easier to develop, transforming the legal profession with more technical lawyers. By design, smart contract cannot be breached: once contracting parties have agreed to be bound by a particular clause, the code's immutability binds them to that clause without leaving them the possibility of a breach; that is, the code defines its own interpretation and enforces the defined rules contained on it without the need of third parties (i.e., self-enforcement). The only way to escape from contractual obligations that the parties no longer want to honor is by including legal provisions into smart contract's code. Over time, law and code may converge, so that infringing the law will be effectively breaking the code: that is, *Lex Cryptographia* is stronger than *Lex Posita* and will become more prevalent.

The strictness of self-enforcement has been much criticized[Lev17, O'H17]: it could be misused and turned against the contracting parties and it doesn't represent the realities of real-world enforcement of contracts. Adding verifiability to smart contracts as proposed in this work prevents all these problems as it allows to check conformity to the specifications from third parties (e.g., governments, regulating bodies, standards). In sum, contracting parties are private lawmakers[Sur12] in control of both the substance and the form of their contractual obligations (i.e., *pacta sunt servanda*), but third parties could intervene to regulate said private agreements by redacting specifications that smart contracts must formally verify against.

7.5 Markets for Smart Contracts

Paradoxically, smart contracts are hardly bought/sold: the inexistence of strong property rights hampers the development of markets on smart contracts; actually, not even companies developing and operating them are being acquired/merged based on the value of their smart contracts since they can easily be reverse engineered and cloned.

Private smart contracts provide strong property rights based on cryptographic techniques which allow for the emergence of markets to trade them. Depending on the cryptographic techniques employed, the following advantageous situations could be considered:

1. The algorithms contained within smart contracts could be traded without disclosing their details when using private function evaluation, homomorphic encryption and/or indistinguishability obfuscation.

2. More practically, when using garbled circuits or secret sharing techniques the source of value will reside on encrypted data processed by the algorithms of private smart contracts, these cryptographic techniques being much more efficient than homomorphic encryption/IO; said encrypted data could be stored on the blockchain, easing the transfer of property of the smart contract.

Traditionally, binary compilation and obfuscation have been used to safeguard the value of software: secure computation techniques offer a provably-secure way to protect the value of even purely open-source software. Furthermore, the ability to safely trade private smart contracts will justify their higher development costs.

7.6 Effects on Currency Competition

Latest analysis on the monetary policy of cryptocurrencies[FVS17] provide insights on the effects of currency competition:

- Monetary equilibrium between private cryptocurrencies will not deliver price stability: profit-maximizing entrepreneurs issuing cryptocurrencies do not have real incentives to provide stable currencies, only to maximise their seigniorage.

- Monetary systems consisting of only private cryptocurrencies in the equilibrium with stable prices do not provide the socially optimum quantity of money: competition between cryptocurrencies is not enough to provide optimal outcomes since entrepreneurs do not internalise the pecuniary externalities by minting additional tokens.

- Unlike private money, government money has fiscal backing because it can tax agents in the economy. But in competition with people willing to hold cryptocurrencies, the implementation of monetary policy in deflationary settings will be significantly impaired since profit-maximizing entrepreneurs

will be unwilling to retire their private currencies and instead choose to increase their issued money.

- Government money could co-exist without intervention in a unique equilibrium with cryptocurrencies if the minted cryptocurrency growths following a predetermined algorithm (e.g., Bitcoin) and the proceedings are used to buy/finance sufficiently productive capital.

Additionally, social efficiency may also be achieved with different cryptocurrencies featuring diverse functionalities that provide market power to their issuers and users: particularly, this is the case of the private and verifiable smart contracts of this paper, since they could be used to provide natural monopolies on the encrypted programs stored on them.

7.7 Token vs. Account-based Cryptocurrencies

Most cryptocurrencies are based on the model of issuing and transacting tokens, using some form of distributed ledger to keep track of the ownership of said tokens (e.g., Bitcoin, Ethereum and Zcash): they're the digital equivalents of cash.

Another unexplored model of cryptocurrency is that of holding funds in accounts at the central bank or in depository institutions, resembling debit cards: in fact, users of Digital Currency Exchanges maintain accounts holding substantial amounts of wealth, but in the form of token-based cryptocurrencies. There are many efficiency gains to be expected from account-based cryptocurrencies: latest macroeconomic models[BK16] show that they would permanently raise GDP by as much a 3% in the USA due to lower bank funding costs, lower monetary transaction costs and lower distortionary taxes; additionally, they would introduce new tools for the central bank to stabilise the business cycle.

The technology required to implement account-based cryptocurrencies is different and more complex than the required for token-based cryptocurrencies: the smart contracts proposed in this paper are a perfect fit for this task, due to their ability to maintain privacy and guarantee their perfect functionality through formal verification techniques.

7.8 Impact on Market Structures

Although disintermediation will be one of the first consequences of the application of cryptographic smart contracts to financial markets, as happened in the past with the introduction of the Internet[CH00], bank and fund concentrations may also increase: information asymmetry is associated with more concentration[Suf07] and information production and its hiding is a valuable activity of banks, that is, opacity has value in itself[DGHO17]. *Ceteris paribus*, it's difficult to estimate the resulting equilibria of the impact of this technology on the level of concentration in the financial industry, given the high number of inter-related variables; nonetheless it's easier to predict that practices like

shadow banking[PAAB13] will increase, as new financial technology accounts for 35% of their growth[BMPS17].

8 Related work

A number of related works using cryptographic techniques are as follows:

- Enigma[ZNP15, Zys16] is coded in Python, a language lacking verification libraries/toolkits and formal semantics so it can't be used for proof-carrying code; its cryptographic protocols are not constant-round, thus settings with some latency will be it excruciatingly slow (i.e., Internet); finally, the only supported distributed ledger is Bitcoin. Later developments show that Enigma is pivoting to a decentralized data marketplace using deterministic and order-preserving encryption[Pro17].
- WYS*[Ras16] offers an elegant solution for verifiable and and secure multi-party computation based on the dependently typed feature of the F* language. Unfortunately, the use of said research language also compromises its real-world adoption; additionally, it's not integrated on any blockchain and does not offer proof-carrying code. Another similar work[ABB+14] uses EasyCrypt to verify Yao's garbled circuits.
- Hawk[KMS+15] focuses on protecting the privacy of transactions using zk-SNARKs: later works like [Solidus[CZJ+17], Confidential Transactions[Gib16], Bolt[GM16]] provide secure transactions at the protocol level with more efficient techniques. It only mentions secure multi-party computation as a tentative way to replace the trusted auction manager, rejecting it as impractical.
- ZeroCash[BCG+14] and Monero[Noe15] provide security for transactions, not smart contracts. Note that secure transactions on blockchains are just a special restricted case of smart contracts enabled with cryptography, but not the other way around.
- Oyente[LCO+16], a symbolic execution tool to formally verify Ethereum smart contracts (EVM).
- Accountable algorithms[KHB+16] propose a commit-and-prove protocol with zk-SNARKs to provide accountability proofs of compliance to legal standards without revealing key attributes of computerized decisions after said decisions have been taken, but not before their execution as is done in the present paper.
- Previous works on combining MPC with Bitcoin[ADMM13b, ADMM13a] use it as support to obtain fairness in MPC, and not better smart contracts.
- Previous projects[MEK+12, ABB+10] designed high-level languages for Zero-Knowledge Proofs of Knowledge but not for Zero-Knowledge Proofs of Proofs, and their languages were restricted and not general purpose.

9 Conclusions, subsequent and future work

The present paper has tackled and successfully solved the problem of improving the privacy, correctness and verifiability of smart contracts, resolving the DAO and Gyges attacks. Examples have been shown to demonstrate its practical viability.

9.1 Subsequent work

The ability to save the state of garbled circuits and restore them at later times is a major improvement, bringing them closer to secret sharing techniques:

- partial garbled circuits[MGBF14]: for each wire value, the generator sends two values to the evaluator, transforming the wire labels of the evaluator to another garbled circuit; depending on its point and permute bit, the value from a previous garbled circuit computation is mapped to a valid wire label in the next computation.
- reactive garbled circuits[NR15]: a generalization of garbled circuits which allows for partial evaluation and dynamic input selection based on partial outputs.
- reusable garbled circuits[GKP+12, Agr16] allow for token-based obfuscation where the code producer provides tokens to code consumers representing rights to execute garbled smart contracts: later constructions[Wan17] improve their concrete efficiency.

Raziel is ongoing development and subject to improvements.

9.2 Future work

Some encrypted smart contracts will be perpetual (e.g., consols): if they ever store any kind of encrypted secure computation (e.g., secret shares, garbled circuits, homomorphic encryptions, IO) eventually there will be the need to update their encrypted contents to upgrade their security level, a concept that has already been considered[LCL+13, ACJ16, GRY17].

MPC and SGX are not mutually exclusive and they will be used jointly to obtain better performance[BBB+16, GMF+16, PST16]. On the other hand, it's difficult to derive realistic threat models and abstractions[BPSW16, PST16, SSL+17] that withstand the latest attacks against SGX [BMD+17, SWG+17, MIE17, WKPK16, XCP15, SLKP17, LSG+16, BCLK17, Swa17, LJJ+17, Cor17].

References

[AACM16] Aysajan Abidin, Abdelrahaman Aly, Sara Cleemput, and Mustafa A. Mustafa. An MPC-based Privacy-Preserving Protocol for a Local Electricity Trading Market. Cryptology ePrint Archive, Report 2016/797, 2016. `http://eprint.iacr.org/2016/797.pdf`.

[ABB+10] Jose Bacelar Almeida, Endre Bangerter, Manuel Barbosa, Stephan Krenn, Ahmad-Reza Sadeghi, and Thomas Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on σ-protocols. Cryptology ePrint Archive, Report 2010/339, 2010. http://eprint.iacr.org/2010/339.

[ABB+14] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Guillaume Davy, François Dupressoir, Benjamin Grégoire, and Pierre-Yves Strub. Verified implementations for secure and verifiable computation. Cryptology ePrint Archive, Report 2014/456, 2014. http://eprint.iacr.org/2014/456.

[ABC16] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on Ethereum smart contracts. Cryptology ePrint Archive, Report 2016/1007, 2016. http://eprint.iacr.org/2016/1007.pdf.

[ABPP15] David W. Archer, Dan Bogdanov, Benny Pinkas, and Pille Pullonen. Maturity and performance of programmable secure computation. Cryptology ePrint Archive, Report 2015/1039, 2015. http://eprint.iacr.org/2015/1039.

[ACJ16] Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. Cryptography with updates. Cryptology ePrint Archive, Report 2016/934, 2016. http://eprint.iacr.org/2016/934.

[ADMM13a] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via the Bitcoin deposits. Cryptology ePrint Archive, Report 2013/837, 2013. http://eprint.iacr.org/2013/837.

[ADMM13b] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure Multiparty Computations on Bitcoin. Cryptology ePrint Archive, Report 2013/784, 2013. http://eprint.iacr.org/2013/784.pdf.

[AF90] Martín Abadi and Joan Feigenbaum. Secure Circuit Evaluation, 1990. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.1130&rep=rep1&type=pdf.

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. Cryptology ePrint Archive, Report 2016/768, 2016. http://eprint.iacr.org/2016/768.

[AGP15] Pablo Daniel Azar, Shafi Goldwasser, and Sunoo Park. How to incentivize data-driven collaboration among competing parties. Cryptology ePrint Archive, Report 2015/178, 2015. http://eprint.iacr.org/2015/178.

[Agr16] Shweta Agrawal. Stronger Security for Reusable Garbled Circuits, General Definitions and Attacks. Cryptology ePrint Archive, Report 2016/654, 2016. http://eprint.iacr.org/2016/654.pdf.

[AHMR14] Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. How to efficiently evaluate RAM programs with malicious security. Cryptology ePrint Archive, Report 2014/759, 2014. http://eprint.iacr.org/2014/759.

[AL11] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly-secure multiparty computation. Cryptology ePrint Archive, Report 2011/136, 2011. http://eprint.iacr.org/2011/136.

[Alg12] Shyamkrishna B. Alganesh. Quasi-Property: Like, But Not Quite Property, 2012. https://www.law.upenn.edu/journals/lawreview/articles/volume160/issue7/Balganesh160U.Pa.L.Rev.1889(2012).pdf.

[All90] Franklin Allen. The market for information and the origin of financial intermediation. *Journal of Financial Intermediation*, 1(1):3–30, March 1990. http://www.sciencedirect.com/science/article/pii/1042-9573(90)90006-2.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. Cryptology ePrint Archive, Report 2013/552, 2013. http://eprint.iacr.org/2013/552.

[AP90] Anat R Admati and Paul Pfleiderer. Direct and Indirect Sale of Information. *Econometrica*, 58(4):901–928, July 1990. https://www.jstor.org/stable/2938355.

[Arr62] Kenneth Arrow. Economic Welfare and the Allocation of Resources for Invention. In *The Rate and Direction of Inventive Activity: Economic and Social Factors*, NBER Chapters, pages 609–626. National Bureau of Economic Research, Inc, December 1962. https://www.nber.org/chapters/c2144.pdf.

[AV17] Abdelrahaman Aly and Mathieu Van Vyve. Practically Efficient Secure Single-Commodity Multi-Market Auctions. Cryptology ePrint Archive, Report 2017/439, 2017. http://eprint.iacr.org/2017/439.pdf.

[AW15] Primavera De Filippi Aaron Wright. Decentralized Blockchain Technology and the Rise of Lex Cryptographia, 2015. http://www.the-blockchain.com/docs/Decentralized%20Blockchain%20Technology%20And%20The%20Rise%20Of%20Lex%20Cryptographia.pdf.

[AY94] James J Anton and Dennis A Yao. Expropriation and Inventions: Appropriable Rents in the Absence of Property Rights. *American Economic Review*, 84(1):190–209, March 1994. https://www.jstor.org/stable/2117978.

[AY02] James J. Anton and Dennis A. Yao. The Sale of Ideas: Strategic Disclosure, Property Rights, and Contracting. *Review of Economic Studies*, 69(3):513–531, 2002. http://hdl.handle.net/10.1111/1467-937X.t01-1-00020.

[AY05] James J. Anton and Dennis A. Yao. Markets For Partially Contractible Knowledge: Bootstrapping Versus Bundling. *Journal of the European Economic Association*, 3(2-3):745–754, 04/05 2005. https://www.jstor.org/stable/40005016.

[BBB+16] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure Multiparty Computation from SGX. Cryptology ePrint Archive, Report 2016/1057, 2016. http://eprint.iacr.org/2016/1057.pdf.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. Cryptology ePrint Archive, Report 2012/095, 2012. http://eprint.iacr.org/2012/095.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. Cryptology ePrint Archive, Report 2014/349, 2014. http://eprint.iacr.org/2014/349.

[BCLK17] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. Rollback and Forking Detection for Trusted Execution Environments using Lightweight Collective Memory, 2017. https://www.ibr.cs.tu-bs.de/users/brandenb/papers/brandenburger_17_dsn.pdf.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation. Cryptology ePrint Archive, Report 2014/404, 2014. http://eprint.iacr.org/2014/404.

[BDLF+16] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. Formal Verification of Smart Contracts: Short Paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, PLAS '16, pages 91–96, New York, NY, USA, 2016. ACM. http://www.cs.umd.edu/%7Easeem/solidetherplas.pdf.

[BDOZ10] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. Cryptology ePrint Archive, Report 2010/514, 2010. http://eprint.iacr.org/2010/514.

[BELO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing Semi-Honest Secure Multiparty Computation for the Internet. Cryptology ePrint Archive, Report 2016/1066, 2016. http://eprint.iacr.org/2016/1066.

[Ben14] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System, 2014. https://arxiv.org/abs/1407.3561.

[BFK+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. Proving the TLS handshake secure (as it is). Cryptology ePrint Archive, Report 2014/182, 2014. http://eprint.iacr.org/2014/182.

[BGW88] Michael BenOr, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract), 1988. https://groups.csail.mit.edu/cis/pubs/shafi/1988-stoc.pdf.

[BK16] John Barrdear and Michael Kumhof. The Macroeconomics of Central Bank Issued Digital Currencies. Bank of England working papers 605, Bank of England, 2016. http://www.bankofengland.co.uk/research/Documents/workingpapers/2016/swp605.pdf.

[BKK+15] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. Students and taxes: a privacy-preserving social study using secure computation. Cryptology ePrint Archive, Report 2015/1159, 2015. http://eprint.iacr.org/2015/1159.

[BLMZ16] Fabrice Benhamouda, Tancrède Lepoint, Claire Mathieu, and Hang Zhou. Optimization of bootstrapping in circuits. Cryptology ePrint Archive, Report 2016/785, 2016. http://eprint.iacr.org/2016/785.

[Blu87] Manuel Blum. How to Prove a Theorem So No One Else Can Claim It. In *In: Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987. http://www.mathunion.org/ICM/ICM1986.2/Main/icm1986.2.1444.1451.ocr.pdf.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: a framework for fast privacy-preserving computations. Cryptology ePrint Archive, Report 2008/289, 2008. http://eprint.iacr.org/2008/289.

[BMD+17] Ferdinand Brasser, Urs Muller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software Grand Exposure: SGX Cache Attacks Are Practical, 2017. https://arxiv.org/pdf/1702.07521.

[BMPS17] Greg Buchak, Gregor Matvos, Tomasz Piskorski, and Amit Seru. Fintech, Regulatory Arbitrage, and the Rise of Shadow Banks, 2017. https://www.gsb.stanford.edu/faculty-research/working-papers/fintech-regulatory-arbitrage-rise-shadow-banks.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The Round Complexity of Secure Protocols, 1990. http://web.cs.ucdavis.edu/~rogaway/papers/bmr90.

[BOGG+90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything Provable is Provable in Zero-Knowledge, 1990. http://crypto.cs.mcgill.ca/~crepeau/COMP647/2007/TOPIC04/BGGHKMR89.pdf.

[BP17] Massimo Bartoletti and Livio Pompianu. An empirical analysis of smart contracts: platforms, applications, and design patterns, 2017. https://arxiv.org/abs/1703.06322.

[BPSW16] Manuel Barbosa, Bernardo Portela, Guillaume Scerri, and Bogdan Warinschi. Foundations of Hardware-Based Attested Computation and Application to SGX. Cryptology ePrint Archive, Report 2016/014, 2016. http://eprint.iacr.org/2016/014.pdf.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of Useful Work. Cryptology ePrint Archive, Report 2017/203, 2017. http://eprint.iacr.org/2017/203.pdf.

[BS14] Marina Blanton and Siddharth Saraph. Secure and Oblivious Maximum Bipartite Matching Size Algorithm with Applications to Secure Fingerprint Identification. Cryptology ePrint Archive, Report 2014/596, 2014. http://eprint.iacr.org/2014/596.pdf.

[BTW11] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. Cryptology ePrint Archive, Report 2011/662, 2011. http://eprint.iacr.org/2011/662.

[But14a] Vitalik Buterin. A Next-Generation Smart Contract and Decentralized Application Platform. Cryptology ePrint Archive, Report 2015/1006, 2014. https://github.com/ethereum/wiki/wiki/White-Paper.

[But14b] Vitalik Buterin. Secret Sharing DAOs: The Other Crypto 2.0, 2014. https://blog.ethereum.org/2014/12/26/secret-sharing-daos-crypto-2-0/.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. Cryptology ePrint Archive, Report 2015/163, 2015. http://eprint.iacr.org/2015/163.

[BZ13] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/642, 2013. http://eprint.iacr.org/2013/642.

[Cac16] Christian Cachin. Architecture of the Hyperledger Blockchain Fabric. https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf, 2016. Accessed: 2016-08-10.

[Cal17] Calctopia. SECCOMP - The Secure Spreadsheet, 2017. https://www.calctopia.com.

[Cas14] Jay Cassano. What are Smart Contracts? Cryptocurrency's Killer App, 2014. http://www.fastcolabs.com/3035723/app-economy/smart-contracts-could-be-cryptocurrencys-killer-app.

[CBB16] Christopher D. Clack, Vikram A. Bakshi, and Lee Braine. Smart Contract Templates: foundations, design landscape and research directions, 2016. https://arxiv.org/pdf/1608.00771.pdf.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086, 2016. http://eprint.iacr.org/2016/086.pdf.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. Cryptology ePrint Archive, Report 2017/279, 2017. http://eprint.iacr.org/2017/279.pdf.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Improving TFHE: faster packed homomorphic operations and efficient circuit bootstrapping. Cryptology ePrint Archive, Report 2017/430, 2017. http://eprint.iacr.org/2017/430.pdf.

[CH00] Eric K. Clemons and Lorin M. Hitt. The Internet and the Future of Financial Services: Transparency, Differential Pricing and Disintermediation. Center for Financial Institutions Working Papers 00-35, Wharton School Center for Financial Institutions, University of Pennsylvania, September 2000. https://ideas.repec.org/p/wop/pennin/00-35.html.

[CHK+11] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces. Cryptology ePrint Archive, Report 2011/257, 2011. http://eprint.iacr.org/2011/257.pdf.

[CLT14] Henry Carter, Charles Lever, and Patrick Traynor. Whitewash: Outsourcing garbled circuit generation for mobile devices. Cryptology ePrint Archive, Report 2014/224, 2014. http://eprint.iacr.org/2014/224.

[CMR17] Brent Carmer, Alex J. Malozemoff, and Mariana Raykova. 5Gen-C: Multi-input Functional Encryption and Program Obfuscation for Arithmetic Circuits. Cryptology ePrint Archive, Report 2017/826, 2017. http://eprint.iacr.org/2017/826.pdf.

[CMTB13] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin Butler. Secure Outsourced Garbled Circuit Evaluation for Mobile Devices. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 289–304, Washington, D.C., 2013. USENIX. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/carter.

[Coa72] Ronald H Coase. Durability and Monopoly. *Journal of Law and Economics*, 15(1):143–149, April 1972. http://dx.doi.org/10.1086/466731.

[Com85] Uniform Law Commission. Uniform Trade Secrets Act with 1985 Amendments, 1985. http://www.uniformlaws.org/shared/docs/trade%20secrets/utsa_final_85.pdf.

[Com16] FIX Trading Community. FIX Protocol Application Layer, 2016. http://www.fixtradingcommunity.org/pg/structure/tech-specs/fix-protocol.

[Cor17] Intel Corp. INTEL-SA-00076: Intel SGX Elevation of Privilege, 2017. https://security-center.intel.com/advisory.aspx?intelid=INTEL-SA-00076&languageid=en-fr.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-Carrying Data and Hearsay Arguments from Signature Cards, 2010. https://people.eecs.berkeley.edu/~alexch/docs/CT10.pdf.

[CT16] Henry Carter and Patrick Traynor. OPFE: Outsourcing computation for private function evaluation. Cryptology ePrint Archive, Report 2016/067, 2016. http://eprint.iacr.org/2016/067.

[CTV13] Stephen Chong, Eran Tromer, and Jeffrey A. Vaughan. Enforcing language semantics using proof-carrying data. Cryptology ePrint Archive, Report 2013/513, 2013. http://eprint.iacr.org/2013/513.

[CZJ+17] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential Distributed Ledger Transactions via PVORM. Cryptology ePrint Archive, Report 2017/317, 2017. http://eprint.iacr.org/2017/317.pdf.

[Dai16] Phil Daian. Analysis of the DAO exploit. Hacking Distributed, 2016. http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit.

[Dam16] Ernesto Damiani. Evaluation and integration and final report on legal aspects of data protection, 2016. https://practice-project.eu/downloads/publications/year3/D31.3-Evaluation-and-integration-and-final-report-on-PU-M36.pdf.

[DBB+15] Gaby G. Dagher, Benedikt Buenz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. Cryptology ePrint Archive, Report 2015/1008, 2015. http://eprint.iacr.org/2015/1008.

[DDN+15] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential benchmarking based on multiparty computation. Cryptology ePrint Archive, Report 2015/1006, 2015. http://eprint.iacr.org/2015/1006.

[DGHO17] Tri Vi Dang, Gary Gorton, Bengt Holmstrom, and Guillermo Ordonez. Banks as Secret Keepers. *American Economic Review*, 107(4):1005–1029, April 2017. https://ideas.repec.org/a/aea/aecrev/v107y2017i4p1005-29.html.

[Dig17a] Digiconomist. Bitcoin Energy Consumption Index, 2017. http://digiconomist.net/bitcoin-energy-consumption.

[Dig17b] Digiconomist. Ethereum Energy Consumption Index, 2017. http://digiconomist.net/ethereum-energy-consumption.

[DPSZ11] I. Damgard, V. Pastro, N.P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011. http://eprint.iacr.org/2011/535.

[Dup08] François Dupressoir. Code and Proof Obfuscation, 2008. http://fdupress.net/files/m2-material/report.pdf.

[EAA11] Andrew W. Lo Emmanuel A. Abbe, Amir Khandani. Privacy-Preserving Methods for Sharing Financial Risk Exposures, 2011. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1962090.

[EL03] Edith Elkind and Helger Lipmaa. Interleaving cryptography and mechanism design: The case of online auctions. Cryptology ePrint Archive, Report 2003/021, 2003. http://eprint.iacr.org/2003/021.

[FKOS13] Mark D. Flood, Jonathan Katz, Stephen J. Ong, and Adam D. Smith. Cryptography and the economics of supervisory information: Balancing transparency and confidentiality, 2013. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2354038.

[FLGR09] Cédric Fournet, Gurvan Le Guernic, and Tamara Rezk. A Security-Preserving Compiler for Distributed Programs: From Information-flow Policies to Cryptographic Mechanisms. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 432–441, New York, NY, USA, 2009. ACM. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/01/a-security-preserving-compiler-for-distributed-programs-ccs09.pdf.

[FPR11] Cédric Fournet, Jérémy Planul, and Tamara Rezk. Information-flow Types for Homomorphic Encryptions. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 351–360, New York, NY, USA, 2011. ACM. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/01/information-flow-types-for-homomorphic-encryptions-ccs11.pdf.

[Fro16] Aymeric Fromherz. Fromherz's smart contracts formalized in Coq. GitHub, 2016. https://github.com/sunblaze-ucb/coq-smart-contract.

[FVS17] Jesús Fernández-Villaverde and Daniel Sanches. Can Currency Competition Work?, 2017. http://economics.sas.upenn.edu/~jesusfv/currency_competition.pdf.

[Gib16] Adam Gibson. An investigation into Confidential Transactions. The Elements Project, 2016. https://github.com/AdamISZ/ConfidentialTransactionsDoc/blob/master/essayonCT.pdf.

[GK83] Mark B. Garman and Steven W. Kohlhagen. Foreign currency option values. *Journal of International Money and Finance*, 2(3):231–237, 1983. https://www.sciencedirect.com/science/article/pii/S0261560683800011.

[GKP+12] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. Cryptology ePrint Archive, Report 2012/733, 2012. http://eprint.iacr.org/2012/733.

[GKS17] Daniel Günther, Ágnes Kiss, and Thomas Schneider. More Efficient Universal Circuit Constructions. Cryptology ePrint Archive, Report 2017/798, 2017. http://eprint.iacr.org/2017/798.pdf.

[GM16] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701, 2016. http://eprint.iacr.org/2016/701.

[GMF+16] Debayan Gupta, Benjamin Mood, Joan Feigenbaum, Kevin Butler, and Patrick Traynor. *Using Intel Software Guard Extensions for Efficient Two-Party Secure Function Evaluation*, pages 302–318. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. http://www.cs.yale.edu/homes/jf/GMFBT-WAHC2016.pdf.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for boolean circuits. Cryptology ePrint Archive, Report 2016/163, 2016. http://eprint.iacr.org/2016/163.

[GMP+08] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS—Secure sessions with handshake and record layer protocols. Cryptology ePrint Archive, Report 2008/251, 2008. http://eprint.iacr.org/2008/251.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 1987. ACM, ACM. https://gnunet.org/sites/default/files/PlayMentalGame1987Goldreich.pdf.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs, 1996. http://class.ece.iastate.edu/tyagi/cpre681/papers/p431-goldreich.pdf.

[Gol87] O. Goldreich. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 182–194, New York, NY, USA, 1987. ACM. http://doi.acm.org/10.1145/28395.28416.

[Gol97] Shafi Goldwasser. Multi-Party Computations: Past and Present, 1997. https://groups.csail.mit.edu/cis/pubs/shafi/1997-podc.pdf.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[GRY17] Paul Grubbs, Thomas Ristenpart, and Yuval Yarom. Modifying an Enciphering Scheme after Deployment. Cryptology ePrint Archive, Report 2017/137, 2017. http://eprint.iacr.org/2017/137.pdf.

[GS03] Joshua S. Gans and Scott Stern. The product market and the market for ideas: commercialization strategies for technology entrepreneurs. *Research Policy*, 32(2):333–350, February 2003. http://www.sciencedirect.com/science/article/pii/S0048-7333(02)00103-8.

[Hir17] Yoichi Hirai. A Next-Generation Smart Contract and Decentralized Application Platform, 2017. https://yoichihirai.com/malta-paper.pdf.

[HIV17] Carmit Hazy, Yuval Ishai, and Muthu Venkitasubramaniam. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. Unpublished, 2017.

[HSR+17] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Deepak Kumar, and Dwight Guth. K Semantics of the Ethereum Virtual Machine (EVM). GitHub, 2017. https://github.com/kframework/evm-semantics.

[Hvi14] Tom Hvitved. A Survey of Formal Languages for Contracts, 2014. https://pdfs.semanticscholar.org/5002/76c957028a65503c4b13214515c07abd5d93.pdf.

[HY16] Carmit Hazay and Avishay Yanai. Constant-round maliciously secure two-party computation in the RAM model. Cryptology ePrint Archive, Report 2016/805, 2016. http://eprint.iacr.org/2016/805.

[iEx17] iEx.ec. The iEx.ec project: blueprint for a Blockchain-based fully distributed cloud infrastructure, 2017. http://iex.ec/wp-content/uploads/2017/04/iExec-WPv2.0-English.pdf.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently, 2003. https://www.iacr.org/cryptodb/archive/2003/CRYPTO/1432/1432.pdf.

[IY88] Russell Impagliazzo and Moti Yung. Direct Minimum-Knowledge Computations, 1988. http://crypto.cs.mcgill.ca/~crepeau/COMP647/2007/TOPIC04/IY89.pdf.

[JE03] S. L. Peyton Jones and J-M. Eber. How to Write a Financial Contract, 2003. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.7885&rep=rep1&type=pdf.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently. Cryptology ePrint Archive, Report 2013/073, 2013. http://eprint.iacr.org/2013/073.

[JKS16] Ari Juels, Ahmed Kosba, and Elaine Shi. The ring of gyges: Investigating the future of criminal smart contracts. Cryptology ePrint Archive, Report 2016/358, 2016. http://eprint.iacr.org/2016/358.

[JT17] Christian Reitwiessner Jason Teutsch. TrueBit: a scalable verification solution for blockchains, 2017. https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf.

[Jut15] Charanjit S. Jutla. Upending stock market structure using secure multi-party computation. Cryptology ePrint Archive, Report 2015/550, 2015. http://eprint.iacr.org/2015/550.

[KHB+16] Joshua A. Kroll, Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson, and Harlan Yu. Accountable Algorithms, 2016. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2765268.

[Kin13] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work), 2013. http://primecoin.io/bin/primecoin-paper.pdf.

[KMO+14] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Bjoern Tackmann, and Daniele Venturi. (De-)constructing TLS. Cryptology ePrint Archive, Report 2014/020, 2014. http://eprint.iacr.org/2014/020.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. http://eprint.iacr.org/2011/272.

[KMR12] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: A system for server-aided secure function evaluation. Cryptology ePrint Archive, Report 2012/542, 2012. http://eprint.iacr.org/2012/542.

[KMS+15] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. Cryptology ePrint Archive, Report 2015/675, 2015. http://eprint.iacr.org/2015/675.

[KS08] Vladimir Kolesnikov and Thomas Schneider. A Practical Universal Circuit Construction and Secure Evaluation of Private Functions, 2008. http://ect.bell-labs.com/who/kolesnikov/papers/UC_FC08.pdf.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. Cryptology ePrint Archive, Report 2016/093, 2016. http://eprint.iacr.org/2016/093.

[Kum16] Ranjit Kumaresan. Privacy-Preserving Smart Contracts, 2016. https://cyber.stanford.edu/sites/default/files/ranjitkumaresan.pdf.

[LCL+13] Kwangsu Lee, Seung Geol Choi, Dong Hoon Lee, Jong Hwan Park, and Moti Yung. Self-updatable encryption: Time constrained access control with hidden attributes and better efficiency. Cryptology ePrint Archive, Report 2013/762, 2013. http://eprint.iacr.org/2013/762.

[LCO+16] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. Cryptology ePrint Archive, Report 2016/633, 2016. http://eprint.iacr.org/2016/633.

[Lem08] Mark A. Lemley. The Surprising Virtues of treating Trade Secrets as IP Rights, 2008. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1155167.

[Lev17] Karen EC Levy. Book-Smart, Not Street-Smart: Blockchain-Based Smart Contracts and The Social Workings of Law. In *Engaging Science, Technology, and Society*, volume 3, pages 1–15, 2017. http://estsjournal.org/article/download/107/61.pdf.

[Lie13] Manuel Liedel. Sichere Mehrparteienberechnungen und datenschutzfreundliche Klassifikation auf Basis horizontal partitionierter Datenbanken, February 2013. https://epub.uni-regensburg.de/27630/.

[LJJ+17] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent ByungHoon Kang. Hacking in Darkness: Return-oriented Programming against Secure Enclaves. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 523–539, Vancouver, BC, 2017. USENIX Association. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk.

[LMA+16] Kevin Lewi, Alex J. Malozemoff, Daniel Apon, Brent Carmer, Adam Foltzer, Daniel Wagner, David W. Archer, Dan Boneh, Jonathan Katz, and Mariana Raykova. 5Gen: A framework for prototyping applications using multilinear maps and matrix branching programs. Cryptology ePrint Archive, Report 2016/619, 2016. http://eprint.iacr.org/2016/619.

[LO15] Steve Lu and Rafail Ostrovsky. Black-box parallel garbled RAM. Cryptology ePrint Archive, Report 2015/1068, 2015. http://eprint.iacr.org/2015/1068.

[LP77] Hayne E Leland and David H Pyle. Informational Asymmetries, Financial Structure, and Financial Intermediation. *Journal of Finance*, 32(2):371–387, May 1977. https://www.jstor.org/stable/2326770.

[LP04] Yehuda Lindell and Benny Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. Cryptology ePrint Archive, Report 2004/175, 2004. http://eprint.iacr.org/2004/175.pdf.

[LSG+16] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing, 2016. https://arxiv.org/abs/1611.06952.

[Mac16] Tanaya Macheel. Banks' Privacy Concerns Shaping Blockchain Vendors' Strategies. American Banker, 2016. https://www.americanbanker.com/news/banks-privacy-concerns-shaping-blockchain-vendors-strategies.

[Mar78] William Margrabe. The Value of an Option to Exchange One Asset for Another. *Journal of Finance*, 33(1):177–186, March 1978. http://onlinelibrary.wiley.com/doi/10.1111/j.1540-6261.1978.tb03397.x/abstract.

[MEK+12] Sarah Meiklejohn, C. Chris Erway, Alptekin Küpçü, Theodora Hinkle, and Anna Lysyanskaya. ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash. Cryptology ePrint Archive, Report 2012/226, 2012. http://eprint.iacr.org/2012/226.

[MGBF14] Benjamin Mood, Debayan Gupta, Kevin R. B. Butler, and Joan Feigenbaum. Reuse It Or Lose It: More Efficient Secure Computation Through Reuse Of Encrypted Values, 2014. http://www.cs.yale.edu/homes/jf/MGBF-CCS14.pdf.

[Mia16] Peihan Miao. Cut-and-choose for garbled RAM. Cryptology ePrint Archive, Report 2016/907, 2016. http://eprint.iacr.org/2016/907.

[MIE17] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. CacheZoom: How SGX Amplifies The Power of Cache Attacks, 2017. https://arxiv.org/abs/1703.06986.

[MJS+14] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing Bitcoin Work for Data Preservation, 2014. http://cs.umd.edu/~amiller/permacoin.pdf.

[MMA+15] Daniel Matichuk, Toby Murray, June Andronick, Ross Jeffery, Gerwin Klein, and Mark Staples. Empirical Study Towards a Leading Indicator for Cost of Formal Software Verification. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 722–732, Piscataway, NJ, USA, 2015. IEEE Press. https://ts.data61.csiro.au/publications/nicta_full_text/8318.pdf.

[Mut90] Shigeo Muto. Resale-proofness and coalition-proof Nash equilibria. *Games and Economic Behavior*, 2(4):337–361, December 1990. http://www.sciencedirect.com/science/article/pii/089982569090004E.

[MZ17] Fermi Ma and Mark Zhandry. Encryptor Combiners: A Unified Approach to Multiparty NIKE, (H)IBE, and Broadcast Encryption. Cryptology ePrint Archive, Report 2017/152, 2017. http://eprint.iacr.org/2017/152.pdf.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. https://bitcoin.org/bitcoin.pdf.

[NL98] George C. Necula and Peter Lee. Safe, Untrusted Agents Using Proof-Carrying Code. In *Mobile Agents and Security*, pages 61–91, London, UK, UK, 1998. Springer-Verlag. http://dl.acm.org/citation.cfm?id=648051.746192.

[Noe15] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. http://eprint.iacr.org/2015/1098.

[NPS99] Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy Preserving Auctions and Mechanism Design. pages 129–139. ACM Press, 1999. http://www.wisdom.weizmann.ac.il/%7Enaor/PAPERS/nps.ps.gz.

[NQ91] Mikio Nakayama and Luis Quintas. Stable payoffs in resale-proof trades of information. *Games and Economic Behavior*, 3(3):339–349, August 1991. http://www.sciencedirect.com/science/article/pii/0899-8256(91)90032-A.

[NR15] Jesper Buus Nielsen and Samuel Ranellucci. Foundations of reactive garbling schemes. Cryptology ePrint Archive, Report 2015/693, 2015. http://eprint.iacr.org/2015/693.

[O'H17] Kieron O'Hara. Smart Contracts-Dumb Idea. In *IEEE Internet Computing*, volume 21, pages 97–101. IEEE, 2017. http://ieeexplore.ieee.org/iel7/4236/7867713/07867719.pdf.

[Ori14] Orisi. Orisi - Distributed Bitcoin Oracles, 2014. http://orisi.org/.

[ORP17] Carlos G. Oliver, Alessandro Ricottone, and Pericles Philippopoulos. Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving NP-complete problems, 2017. https://arxiv.org/pdf/1708.09419.

[PAAB13] Zoltan Pozsar, Tobias Adrian, Adam B. Ashcraft, and Hayley Boesky. Shadow Banking, 2013. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2378449.

[PE16] Jack Pettersson and Robert Edström. Safer smart contracts through type-driven development, 2016. http://publications.lib.chalmers.se/records/fulltext/234939/234939.pdf.

[Pop04] J. W. Pope. Proving a Theorem in Zero-Knowledge, 2004. http://euler.nmt.edu/%7Ebrian/students/pope.pdf.

[Pro17] Enigma Project. Towards a Decentralized Data Marketplace - Part 2, 2017. https://blog.enigma.co/towards-a-decentralized-data-marketplace-part-2-1362c8e11094.

[PST16] Rafael Pass, Elaine Shi, and Florian Tramer. Formal Abstractions for Attested Execution Secure Processors. Cryptology ePrint Archive, Report 2016/1027, 2016. http://eprint.iacr.org/2016/1027.pdf.

[PV16] Marie Paindavoine and Bastien Vialla. Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption, 2016. https://pdfs.semanticscholar.org/aaff/0e7673183181c7f4a241d31f7079da1a9573.pdf.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. http://eprint.iacr.org/2005/187.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978. people.csail.mit.edu/rivest/RivestAdlemanDertouzos-OnDataBanksAndPrivacyHomomorphisms.pdf.

[Ras16] Aseem Rastogi. Language-based Techniques for Practical and Trustworthy Secure Multi-party Computations, 2016. http://drum.lib.umd.edu/bitstream/handle/1903/18541/Rastogi_umd_0117E_17248.pdf?sequence=1&isAllowed=y.

[Ris16] Michael Risch. Hidden in Plain Sight, 2016. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2761100.

[RS97] Jerome H. Reichman and Pamela Samuelson. Intellectual Property Rights in Data?, 1997. https://www.law.berkeley.edu/php-programs/faculty/facultyPubsPDF.php?facID=346&pubID=66.

[RT84] Ram T. S. Ramakrishnan and Anjan V. Thakor. Information Reliability and a Theory of Financial Intermediation. *Review of Economic Studies*, 51(3):415–432, 1984. http://hdl.handle.net/10.2307/2297431.

[SA17] International Swaps and Derivatives Association. FpML Standard, 2017. http://www.fpml.org/the_standard/current/.

[Sch12] Lauren Henry Scholz. Privacy as Quasi-Property, 2012. https://ilr.law.uiowa.edu/print/volume-101-issue-3/privacy-as-quasi-property/.

[SCO18] SCOTUS. International News Service v. Associated Press, 248 U.S. 215 (1918), 1918. https://supreme.justia.com/us/248/215/case.html.

[SGR85] Silvio Micali Shafi Goldwasser and Charles Rackoff. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract), 1985. https://groups.csail.mit.edu/cis/pubs/shafi/1985-stoc.pdf.

[Sha79] Adi Shamir. How to Share a Secret, 1979. https://cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf.

[SJA+14] Mark Staples, Ross Jeffery, June Andronick, Toby Murray, Gerwin Klein, and Rafal Kolanski. Productivity for Proof Engineering. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 15:1–15:4, New York, NY, USA, 2014. ACM. http://dl.acm.org/citation.cfm?id=2652551.

[SLKP17] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs, 2017. https://www.internetsociety.org/sites/default/files/ndss2017_07-2_Shih_paper.pdf.

[SSL+17] Pramod Subramanyan, Rohit Sinha, Ilia Lebedev, Srinivas Devadas, and Sanjit Seshia. A Formal Foundation for Secure Remote Execution of Enclaves. Cryptology ePrint Archive, Report 2017/565, 2017. http://eprint.iacr.org/2017/565.pdf.

[SSW17] Peter Scholl, Nigel P. Smart, and Tim Wood. When It's All Just Too Much: Outsourcing MPC-Preprocessing. Cryptology ePrint Archive, Report 2017/262, 2017. http://eprint.iacr.org/2017/262.pdf.

[STM16] Pablo Lamela Seijas, Simon Thompson, and Darryl McAdams. Scripting smart contracts for distributed ledger technology. Cryptology ePrint Archive, Report 2016/1156, 2016. http://eprint.iacr.org/2016/1156.pdf.

[Suf07] Amir Sufi. Information Asymmetry and Financing Arrangements: Evidence from Syndicated Loans. *The Journal of Finance*, 62(2):629–668, 2007. http://dx.doi.org/10.1111/j.1540-6261.2007.01219.x.

[Sur12] Harry Surden. Computable Contracts, 2012. http://lawreview.law.ucdavis.edu/issues/46/2/Articles/46-2_Surden.pdf.

[Swa17] Yogesh Swami. SGX Remote Attestation is not Sufficient. Cryptology ePrint Archive, Report 2017/736, 2017. http://eprint.iacr.org/2017/736.pdf.

[SWG+17] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clementine Maurice, and Stefan Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks, 2017. https://arxiv.org/abs/1702.08719.

[SZ13] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. Financial Cryptography 2013, 2013. http://fc13.ifca.ai/proc/8-3.pdf.

[Sza97] Nick Szabo. The Idea of Smart Contracts. Nick Szabo's Papers and Concise Tutorials, 1997. https://web.archive.org/web/20150812055200/http://szabo.best.vwh.net/idea.html.

[USCoA94] First Circuit. United States Court of Appeals. Data Gen. Corp. v. Grumman Systems Support Corp., 1994. www.leagle.com/decision/1994118336F3d1147_11018.

[UU91] USPTO and USCO. Patent-Copyright Laws Overlap Study, 1991. https://cdn.patentlyo.com/media/2017/05/1991-Patent-Copyright-Overlap-Study.pdf.

[Val76] Leslie G. Valiant. Universal Circuits (Preliminary Report). In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 196–203, New York, NY, USA, 1976. ACM.

[Vic61] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961. http://dx.doi.org/10.1111/j.1540-6261.1961.tb02789.x.

[Wan17] Xu An Wang. *Toward Construction of Efficient Privacy Preserving Reusable Garbled Circuits*, pages 81–92. Springer International Publishing, Cham, 2017. http://dx.doi.org/10.1007/978-3-319-49109-7_8.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying Computations Without Reexecuting Them. *Commun. ACM*, 58(2):74–84, January 2015. http://doi.acm.org/10.1145/2641562.

[WC14] John Ross Wallrabenstein and Chris Clifton. Privacy Preserving Tâtonnement - A Cryptographic Construction of an Incentive Compatible Market, 2014. http://fc14.ifca.ai/papers/fc14_submission_2.pdf.

[WCS14] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit ORAM: On tightness of the goldreich-ostrovsky lower bound. Cryptology ePrint Archive, Report 2014/672, 2014. http://eprint.iacr.org/2014/672.

[WKPK16] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves, 2016. https://www.ibr.cs.tu-bs.de/users/weichbr/papers/esorics2016.pdf.

[WMK16a] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016. https://github.com/emp-toolkit.

[WMK16b] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster Secure Two-Party Computation in the Single-Execution Setting. Cryptology ePrint Archive, Report 2016/762, 2016. http://eprint.iacr.org/2016/762.pdf.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014. http://gavwood.com/paper.pdf.

[WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. Cryptology ePrint Archive, Report 2017/030, 2017. http://eprint.iacr.org/2017/030.pdf.

[WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-Scale Secure Multiparty Computation. Cryptology ePrint Archive, Report 2017/189, 2017. http://eprint.iacr.org/2017/189.pdf.

[WSC+07] Jim Woodcock, Susan Stepney, David Cooper, John Clark, and Jeremy Jacob. The Certification of the Mondex Electronic Purse to ITSEC Level E6. *Form. Asp. Comput.*, 20(1):5–19, December 2007. http://www-users.cs.york.ac.uk/~jac/PublishedPapers/TheCertificationOfMondex2007.pdf.

[XCP15] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 640–656, Washington, DC, USA, 2015. IEEE Computer Society. https://www.cs.utexas.edu/~yxu/files/xu15oakland.pdf.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract), 1982. https://research.cs.wisc.edu/areas/sec/yao1982-ocr.pdf.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract), 1986. https://pdfs.semanticscholar.org/7bef/c5470606b8ca8e959db4f97daba127c411dd.pdf.

[ZH17] Jack Petterson Zackary Hess, Yanislav Malahov. Aeternity blockchain: The trustless, decentralized and purely functional oracle machine, 2017. https://blockchain.aeternity.com/%C3%A6ternity-blockchain-whitepaper.pdf.

[ZNP15] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized Computation Platform with Guaranteed Privacy, 2015. http://www.enigma.co/enigma_full.pdf.

[ZWR+16] Samee Zahur, Xiao Shaun Wang, Mariana Raykova, Adria Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting Square-Root ORAM: Efficient Random Access in Multi-Party Computation, 2016. http://oblivc.org/docs/sqoram.pdf.

[Zys16] Guy Zyskind. Efficient secure computation enabled by blockchain technology, 2016. https://dspace.mit.edu/bitstream/handle/1721.1/105933/964695278-MIT.pdf.

A Outsourcing Secure Computations for Cloud-based Blockchains

Theorem 4. *(Outsourcing for Cloud-Based Blockchains). Assuming secure channels between the parties $E_1, E_2, ..., E_N$ and the nodes N_i of the blockchain B, Protocol B.2 securely realises Functionality B.1 against static corruptions in the semi-honest security model with the garbling scheme satisfying privacy, obliviousness and correctness.*

Proof. Two cases must be analyzed: corruption at the node N_E executing the secure computation and corruption of the parties.

Corruption at node N_E. The simulator $Simul_{N_E}\left(1^\lambda, result_{N_E}\right)$, where $result_{N_E}$ stores the $KeyExchange$, $SendPrivateParameters$ and $SECCOMP$ run by each party $E_1, E_2, ..., E_N$:

- $KeyExchange\left\langle E_i\left(1^\lambda\right), N_E\right\rangle$: sample a public key pk_i for party E_i.
- $SendPrivateParameters\left\langle E_i, N_E\right\rangle$: sample a vector of random values $c_i = \left(\left(X_{i1}^{x_i[1]}, ..., X_{il}^{x_i[l]}\right), n_i\right)$.

- $SECCOMP\,\langle E_k, N_E\rangle\,(SC)$: a simulated garbled circuit and garbled values are computed. Then, the simulator computes one entry of the pivot tables by encrypting random values and the other entry by encrypting each garbled value with the random values generated in the $SendPrivateParameters$ phase:
  - A simulator of garbled circuits generates a simulated garbled circuit GC_{SC} and garbled values w_{jl} for each party E_j and index l of the length of $\overrightarrow{x_i}$.
  - Compute $Enc_{s_{jl}}(0)$ with random key s_{jl} and $Enc_{X_{jl}^{x_j[l]}}(w_{jl})$ and save them into pivot table $P_q[j,l]$ in random order.
  - Return the pivot tables P_q for each E_j, the garbled circuit GC_{SC} and values w_{jl}.

We prove using hybrid arguments that the view generated by $Simul_{N_E}$ is indistinguishable from the view obtained by N_E in the real world:

H_0. The real world view is computed using the real inputs of the parties and according to the original protocol.

H_1^i. What is different between H_1^i and H_1^{i+1} is that for each E_i in H_1^i the inputs are encoded using random values: that is, $X_{il}^{x_i[l]} = rand_{il}$. It would be possible to obtain a distinguisher for the pseudorandomness of PRF assuming a distinguisher between H_1^i and H_1^{i+1}. The values $X_{il}^{x_i[l]}$ will be computed with an oracle by the reduction: the view will be distributed as in game H_1^i if the oracle is using a random function; otherwise, it will be distributed as in game H_1^{i+1} if the oracle is a pseudo-random function. Thus, a contradiction will be reached since any adversary distinguishing H_1^i from H_1^{i+1} with non-negligible probability will be able to break the pseudo-randomness with the same probability.
The sequence of hybrid games starts with H_1^1 where random values encode inputs of party E_i, to H_1^n where random values from parties $E_1 ... E_N$ encode inputs of the parties.
Finally, note that the hybrid $H_0 = H_1^0$ corresponds to the case where all inputs are pseudo-random values, while the last game H_1^n corresponds to the case in which random values encode all inputs.

$H_2^{i,j,l}$. In this hybrid, the call $SECCOMP$ by party E_i computes the pivot table $P_q[j,l] = Enc_{s_{jl}^{x_j[l]}}(0), Enc_{s_{jl}^{x_j[l]}}\left(w_{jl}^{x_j[l]}\right)$, that is, only one garbled value per wire is encrypted: therefore, a contradiction will be reached since an adversary distinguishing between $H_2^{i,j,l}$ and $H_2^{i,j,l+1}$ with non-negligible probability can be reduced to a distinguisher for the indistinguishability under chosen-plaintext attacks (IND-CPA) of the encryption scheme.
The sequence of hybrid games starts with $H_2^{1,1,l}$ where the pivot table of party E_i is encoded with only one garbled value per wire, to $H_2^{n,n,l}$ where all pivot

tables of parties $E_1 ... E_N$ are encoded with only on garbled value per wire.

H_3^j. The simulator $Simul_{N_E}$ computes the pivot tables and a simulator of garbled circuits is used instead of the real garbling scheme. It would be possible to obtain a distinguisher for garbled circuits assuming a distinguisher between $H_2^{n,n,l}$ and H_3^j: the inputs to the garbled circuits circuit will be computed by an oracle by the reduction as in $H_2^{n,n,l}$, to get garbled values and garbled circuits. Finally, we reach the ideal world with hybrid H_3^n, finishing the proof that started with the real world game H_0.

Corruption of the parties. Assuming a single party E_i is corrupted and secure channels between the parties and N_E, the simulator $Simul_{E_i}$ is defined as:

- $KeyExchange \left\langle E_i \left(1^\lambda\right), N_E \right\rangle$: a public key pk_i is chosen by party U_i.
- $SendPrivateParameters \left\langle E_i, N_E \right\rangle$: using the honest version for $\overrightarrow{x_i}$, honestly compute values $X_{il}^{\overrightarrow{x_i}[l]} = PRF_{k_S}\left(\overrightarrow{x_i}, l, n_i\right)$.
- $SECCOMP \left\langle E_k, N_E \right\rangle (SC)$: honestly compute garbled circuit and pivot tables. For the result, select the garbled values according to Functionality B.1 called with value x_i.

Note that the method to compute the result is the only difference in computing the view of E_i: while the evaluation of the garbled circuit is required in the real world, in the ideal world the correct resulting garbled values are chosen by the simulator $Simul_{E_i}$ using the result of Functionality B.1. Therefore, the indistinguishability of the view of E_i is due to the secure channels: the case for more corrupted parties trivially follows from this argument. □