Alexander Lee (ahl256) and Mitchell Lloyd (mel652)
Applied Cryptography and Network Security, Spring 2022
4/15/2022

Oracle Security in Public Blockchain Systems

Introduction

Since the introduction of Bitcoin in 2008, public blockchain systems have steadily garnered increased attention in both the technology sector and among the general public. While proposals for the design of these systems have evolved over the years, the essential nature of a public blockchain as an append-only transaction ledger maintained by a decentralized network of pseudonymous peers who are rewarded for continued creation of ledger entries (in a unit generally termed "cryptocurrency") has remained fairly constant. Following the launch of the Ethereum public blockchain in 2015, many designs have followed a "generalized computation" model, where users may store general-purpose application logic in the state of the system, and transactions that are appended to the blockchain may include calls to this logic.[1]

In the general-purpose computation model, applications deployed to a blockchain can typically read (and possibly write) data stored in either the state or the blockchain log associated with the system itself, but data that is stored outside of the blockchain system is not directly accessible to these applications. Most digital data in existence are naturally recorded outside of public blockchains, including many data points to which on-chain applications may wish to refer. One popular use case for such external data points is the settlement of financial agreements, whose logic (and settlement "currencies") may be encoded in the blockchain system, but whose reference prices exist outside of the system.[2] Due to the demand for such external (or "off-chain") data in applications deployed to public blockchains, various software systems have been devised to query such data on demand and deliver them to the application logic that resides in the blockchain system; these supplemental systems fall under the broad category of "oracles." In the remainder of this paper, we shall assume that the reader is at least familiar with general-purpose public blockchains as described above, and we will focus specifically on oracle systems and their associated security issues.[3]

---

[1] Bitcoin and systems that copied its design prior to Ethereum use a (deliberately) more limited computational model that is focused primarily on the control of amounts of bitcoin cryptocurrency on the ledger. Ethereum and later systems typically allow "arbitrary" computation (up to a limit, to prevent denial-of-service attacks on transaction processing nodes) and often feature user-created assets (typically called "tokens") in addition to the "native" cryptocurrency of the blockchain system.

[2] Because public blockchain systems incur a fee for transactions, many applications deployed to these systems are themselves financial in nature – paying a fee to perform a computation that could be performed much more cheaply on a different sort of computational platform typically only makes sense if something is "at stake" in that computation, such as the movement of a financial instrument like a cryptocurrency or "token" recorded on a blockchain.

[3] For more information on the general-purpose blockchain system model, the interested reader is referred to Antonpoulos and Wood's "Mastering Ethereum" [1] for a comprehensive overview of the titular system, whose logic applies to several other systems introduced in its wake.

The problem of oracle security

While the design of individual oracle systems varies, typically they comprise at least two major components: one portion of the oracle resides on-chain (as its own application logic deployed to a public blockchain), which other applications deployed to the same blockchain system may interact with directly. The other portion of the oracle system resides outside of the blockchain and is typically responsible for polling for on-chain requests, obtaining the requested data, and dispatching the data via a callback through the on-chain portion of the oracle to the original requestor. A simplified schematic of this flow is presented below:

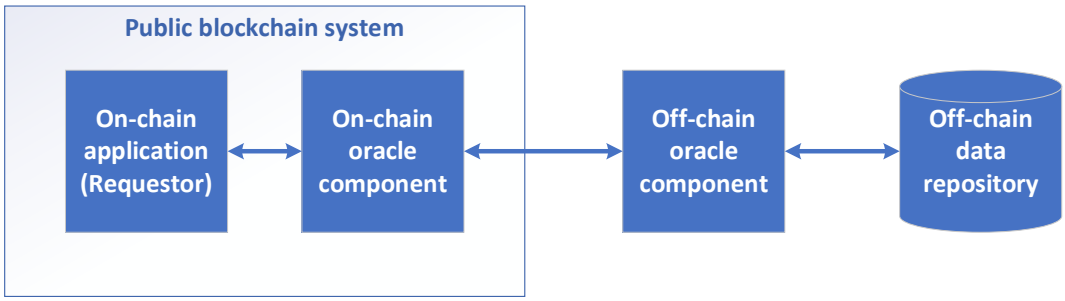

Figure 1: Schematic data flow between on-chain and off-chain components of an oracle request.

The security of oracle systems thus relies on at least four critical points: the application security of the on-chain portion of the system, the application security of the off-chain portion, the security of the interconnect between these two portions, and the security of the interconnect between the off-chain portion of the oracle and the repository containing the requested data (which is most likely not co-located with the off-chain oracle application). The provenance of the data itself represents yet another security concern, but at a certain point this becomes something of a philosophical rather than a technical matter; for purposes of this report, we consider the data in its source repository to be "secure" and in fact the data that the requestor on-chain wishes to obtain.[4] Similarly, we do not address the security of the requestor application, as this is (while very important), an orthogonal concern to the security of the oracle. We now explore each of the four identified critical components in detail.

The on-chain oracle component has the advantage of being able to rely on the security of the blockchain system itself for transaction verification.[5] However, the application logic itself is typically not "validated" in any way by the blockchain system itself, and may contain bugs and/or malicious backdoors. By their public nature, public blockchain systems enable auditing of any deployed code, but this code may only be easily obtainable in a binary format if the oracle system creator does not openly provide their source code, and decompiling and reverse engineering this code to scrutinize its security may be a tall order in practice. Assuming that the application logic itself is not disastrously buggy, of particular concern for this component are the intent of that logic (for example, does it perform any signature validation on data fetched from outside the blockchain, or simply act as a pass-through mechanism?) and the ability of

[4] A technical solution to the question of data provenance is to use data signed by a provider, with an on-chain facility for digital signature verification, as suggested in Arief, Algya, and Lee, 2019 [2] (and elsewhere); however, if two nominally "equally-valid" signatories report different data (as may happen in the real world with financial data providers), it is difficult to determine the "correctness" of the data in the eyes of its user.

[5] Typically, in a system like Ethereum, all transactions must be digitally signed and signatures must verify before transactions can fully process.

an operator to subvert the mechanism in some way. This lattermost is in fact a cross-cutting concern that touches most or all of the four critical components: an unscrupulous operator may wish to (or be bribed to) cause the oracle system to report false data that advantages one party in a particular financial transaction. As will be discussed in greater detail in the following section, various approaches to this problem have been proposed, but it appears difficult to devise a defense against "exit scams" in which a pseudonymous oracle operator elects to burn any reputation that accrued to their identity in favor of a large one-time payout. Unfortunately, this appears to be an area where cryptography alone may be of limited use, though if an oracle application performs digital signature validation on-chain and there is provably no backdoor for the operator to use subversively, this risk may be greatly ameliorated.[6]

The off-chain oracle component presents a potentially larger security risk than the on-chain portion, primarily because it need not be deployed "in public" such that its (compiled) source is subject to scrutiny. An additional concern regarding the code is that it is easier for an attacker or unscrupulous operator to replace with malicious logic: for many public blockchain systems, deployed application code is "immutable" and cannot easily be meddled with in an undetectable manner; however, an off-chain oracle component is essentially opaque to on-chain applications requesting oracle services, and it would be difficult to detect if changes were made to such a component.[7] Many of the same issues discussed above related to on-chain application security (soundness of logic, exit scams) apply to this component as well, and because the off-chain component is almost inevitably connected to the public internet to perform its services, it is additionally subject to denial-of-service network attacks.[8] While the on-chain portion of the oracle system may be considered relatively secure among the four critical portions, the off-chain oracle application is arguably the least secure due to its opacity and its own vulnerability to network-based attacks, even if the operator is acting in good faith.

The interconnection mechanism between the on-chain and off-chain components of the oracle is subject more to questions of network security than application security, depending on the design of the system; for purposes of this discussion, we assume that there is no "middleware" (or that it can be grouped together as part of the off-chain oracle component), and that the off-chain component must

---

[6] Even in such a scenario, the data signatory themselves might be subject to corruption, and for example might be bribed into affixing their valid (and currently-trusted) digital signature to a false piece of data.

[7] In systems like Ethereum, this immutability is by design: the "address" at which a deployed application resides in the state of the system is determined by the cryptographic hash of its source code, and thus any given application may only ever reside at one address. A way to circumvent this is to use a so-called "forwarding pattern" where an application is deployed at a fixed address, but uses mutable storage in its design, and the "forwarding address" for dispatching calls may be updated. However, as the system is public, any changes made to the forwarding address would be publicly visible to anyone watching for them.

[8] While public blockchains themselves are theoretically also subject to such attacks, the fee-for-computation model tends to impose a high cost for such an attack: in several such blockchains including Bitcoin and Ethereum, there is a market for "block space" (that is, the ability to have your transaction potentially recorded in the next block of transactions appended to the ledger), and in order to launch a denial-of-service attack on such a system, the attacker would have to out-bid all other legitimate transactions in this fee market, which could become very expensive to sustain. This being said, public blockchains do tend to have fairly limited throughput in terms of transactions per unit of time, so congestion can easily be a real factor that might in certain circumstances affect the behavior of an on-chain component that needed to deliver data in a timely fashion. Good on-chain application design should be aware of such potential congestion and ideally never try to impose real-time constraints on the system that it will be unable to deliver (the probabilistic nature of appending transaction batches, in systems that use proof-of-work-based consensus, poses the same problem for applications with real-time expectations).

dispatch data to the on-chain component by making a blockchain transaction. Viewed in this fashion, the security of this interconnection partially collapses to the security of blockchain transactions in general, as the transaction will not be accepted without a valid digital signature affixed. However, the blockchain system itself is not (and effectively cannot be) responsible for the "metaphysical validity" of the data, which is a problem affecting digital signatures generally.[9] In addition, due to the non-obfuscated nature of transactions on most general-purpose public blockchain systems, transactions that invoke functionality of a specific oracle system may be easily identifiable as such, and are thus more "targetable" if an entity wished to deny a transaction from being appended to the log.[10]

The "last-hop" interconnection between the off-chain oracle component and the data repository from which it obtains the requested data is almost purely subject to network security concerns. Many oracle services allow requestors to specify essentially arbitrary URL endpoints for the data that they request, meaning the connection to that endpoint is subject to attack (and the URL of the endpoint can be discovered by an attacker simply by observing public blockchain transaction traffic, by running their own node for example). A sophisticated attacker could use techniques such as DNS poisoning to route requests to an alternative endpoint under their control, and even if the off-chain oracle component uses secure connections to endpoints, the attacker could potentially obtain a valid SSL certificate for the endpoint before the attack was made (after the attack, the certificate would presumably be discovered to have been used for a fraudulent purpose and would be revoked, but the attack may have succeeded by that point). Because the data repository is very likely accessible on a public network, it is also subject to standard denial-of-service attacks (which are more difficult to perform against a public blockchain system for sustained periods of time, as discussed earlier). In general, this interconnection may be considered more vulnerable to a variety of attacks than the connection between the on-chain and off-chain components of the oracle system, and leaving aside the issue of an unscrupulous oracle operator, may arguably be considered the most vulnerable component of the setup illustrated in Figure 1.

While the security concerns described above are not a comprehensive list of issues facing oracle systems, they are nevertheless illustrative of the broad range of issues facing blockchain-based applications that wish to make use of oracle services. We next examine some proposed and existing solutions to certain of these problems, before introducing a novel solution for certain classes of financial applications.

---

[9] Cryptography can make certain guarantees about mathematical properties of a digital signature over a piece of data, but leaves the question of the "meaning" of such data to philosophers (or perhaps lawyers and judges, as the case may be).

[10] This possibility is counterbalanced by the nature of transaction validation and appending of new log entries on most public blockchain systems: in proof-of-work-based systems, anyone may run a validating node and have a chance of adding new transactions proportional to the computational power they bring to the network overall, which may be small, but is nonzero. If such a network is relatively "decentralized" by some suitable measure, an attacker would have to coopt many validating nodes simultaneously and for a sustained period in order to try to censor a specific transaction, or transactions bound for a specific destination application on the blockchain. Such an attack would be much more difficult to perform than a "standard" network-based denial-of-service attack. Systems that use alternative validation mechanisms such as proof-of-stake still typically exhibit some randomness in the selection of validators for any given round in which the log will be appended to, making such an attack difficult regardless of the difference in validation mechanism.

Current approaches to securing oracles for public blockchains

We note at the outset of this section that securely connecting oracle systems to public blockchains is not a "solved problem," and that the approaches that various service providers have pursued to date are far from comprehensive. However, we can still identify several stylized categories of approaches to oracle security, illustrated in Figure 2 below. It is important to note that these approaches are not necessarily mutually exclusive in practice, and layering several of them for the same application could potentially provide enhanced security for the oracle system overall. We describe each in some detail below, and note instances where applications are likely to layer these approaches in practice.

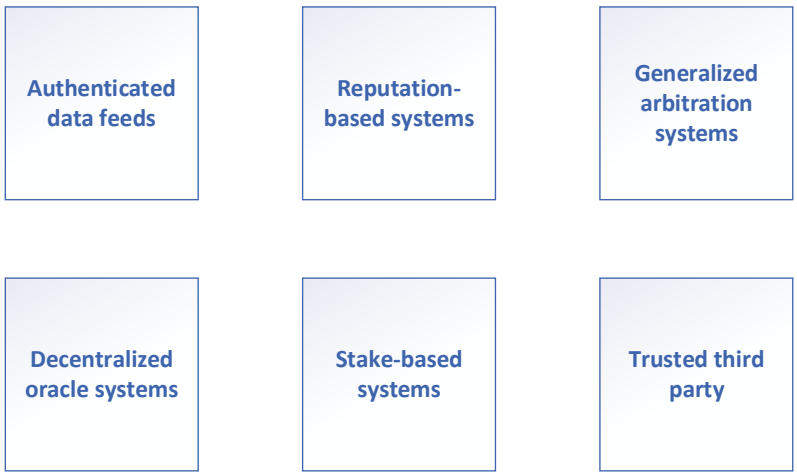

Figure 2: Stylized approaches to oracle security for public blockchain systems.

Authenticated data feeds for public blockchain systems were formally proposed in 2016 in the Town Crier whitepaper [3], and multiple projects have implemented systems in line with this design (e.g., [4], [5]). Town Crier specifically proposes an end-to-end authentication pipeline connecting on-chain applications to desired data (which must be available on HTTPS-enabled websites), and proposes using trusted execution environments (TEEs) for verifying the computations performed off-chain (e.g., attesting via a digital signature to having fetched a particular data point from a particular URL between two Unix timestamps before delivering it back to the blockchain where the requestor application resides). While conceptually appealing, such designs still face issues noted above related to fraudulent SSL certificates on HTTPS-enabled websites, and attacks against TEEs themselves.[11]

Decentralized oracle systems are in one sense an orthogonal approach to authenticated data feeds: rather than trusting a single chain of authentication from the data source back to the requestor, a decentralized oracle system can allow a request to be made for the same data through multiple service providers simultaneously, with some mechanism for combining and filtering the received responses. Thus, even if some small percentage of oracle service providers are untrustworthy, a useful data point may still be obtained from outside of a blockchain for use in a blockchain-based application. While the individual oracle systems used in a decentralized query may still be coopted in the manners discussed earlier, spreading out the potential target in this manner nominally increases the security of the system. However, decentralized oracle systems also open up new attack vectors, particularly "Sybil attacks" in which a single

[11] For example, see Nilsson, Bideh, and Brorsson [6] for a survey of attacks against Intel SGX, the TEE which the original Town Crier paper used to demonstrate implementation of the system.

entity masquerades as multiple distinct entities. If the target of such an attack does not realize that all of the oracle nodes through which they are requesting a service are in fact controlled by the same entity, the attacker may cause all of the nodes to report the same (false) value. For this reason, decentralized oracle systems are often combined in practice with additional security measures that attempt to mitigate the risk of Sybil attacks.

Reputation- and stake-based systems can help add security to a decentralized oracle model by presenting a would-be attacker with a penalty in the event that they attempt to defraud a requestor. In a system with known identities, reputation "alone" may be sufficient to deter attacks: assuming that oracle service providers are paid a fee for their service, they might be hesitant to destroy a carefully-built reputation that allows them to earn such fees. [12] However, many decentralized systems are pseudonymous, and reputations may become harder to validate accordingly. To circumvent this issue somewhat in such a pseudonymous setting, decentralized oracle service providers may instead stake something of direct monetary value (rather than their "name") on their service, with some mechanism for punishing cheaters by destroying or reapportioning this staked value. Chainlink [4] has proposed using authenticated data feeds and a stake-based decentralized oracle network in concert for extra security. UMA [7] illustrates one approach to penalizing an oracle service provider that provides incorrect data by allowing a dispute to be opened, and if a decentralized panel of governing entities votes in favor of the disputer, a bond posted by the oracle service provider will be taken from them.[13] However, none of these approaches can in theory prevent a pseudonymous oracle service provider from engaging in an "exit scam" attack: even if such an entity has built a valuable (pseudonymous) reputation or has staked considerable monetary value, it is always possible that the reward that would accrue to the attacker for destroying their reputation or stake would far outweigh the value of the latter. This remains an unsolved problem for pseudonymous settings in general.

Generalized arbitration systems attempt to offer dispute resolution on-chain (similar to UMA's resolution mechanism described above), but in a manner that is separated from the oracle services themselves. Kleros [8] proposes such a system that on-chain applications can interface with to provide dispute resolution regarding transactions pertinent to that particular application. This approach sidesteps the question of oracle reputation (and in some sense, oracle security generally) by allowing on-chain applications to facilitate arbitration of situations arising from incorrect (or disputed) data arising from any oracle system, no matter its security status. However, it reintroduces the decentralized reputation (and/or staking) concerns discussed above (as these systems are typically decentralized and controlled by on-chain, pseudonymous voting), now in the context of the arbitration system rather than the oracle service. Thus, such systems may ultimately "relocate" security problems for oracles, rather than solving them.

Finally, trusted third parties (TTPs) can entirely circumvent the question of oracle security, at the cost of allowing the TTP to potentially break that trust under certain circumstances.[14] In general, public

[12] Of course, reputation truly in a vacuum may be insufficient; the ability of a victim to subject an identified attacker to a justice system in some shared jurisdiction may be even more important here. We treat "reputation" as covering both the desire to uphold "one's good name" and also to keep said name from being summoned before a court.

[13] The astute reader may wonder who these decentralized governing entities are, and whether they can be trusted not to engage in Sybil attacks themselves; such readers are justified in their concerns, and this remains another unsolved problem in the oracle security space.

[14] The earlier discussion of exit scams is pertinent here as well, and trusted third parties that are known to the requestor may additionally attempt to economically negotiate with the requestor directly in certain cases.

blockchain systems tend to encourage the elimination of TTPs where possible, which has likely spurred the development of various oracle systems that attempt to decentralize the task of data provision, or to introduce cryptographic verification mechanisms in place of trust. If a third party were truly "trusted" (for example, the original purveyor of some piece of off-chain data), a simple mechanism proposed in [2] would be for the purveyor to digitally sign data points, which could then be brought on-chain by any expedient (and potentially "insecure") means, as long as signature verification logic were available on-chain as well. One notable concern related to TTPs is recognizing when they exist: some systems, such as MakerDAO [9] purport to use decentralized oracles (in the case of MakerDao, as part of their "stablecoin" service offering), but the identities of these oracles are unknown to anyone except the Maker Foundation and are not published, meaning that the Foundation itself is effectively a TTP mediating the data flowing through these oracles.

A novel solution: economically rational settlement

Given that the approaches discussed in the previous section do not satisfactorily solve the oracle problem in the general case, we propose an economic mechanism for avoiding the need to rely on (the security of) an oracle system in cases where distribution of funds is contingent upon the data fetched from outside of a blockchain system by an oracle service, e.g., financial derivative contracts.[15] We term this mechanism economically rational settlement (ERS) as it is premised on the economic notion of rational utility-maximizing agents in a game-theoretic setting.[16] Given certain economic assumptions, transaction counterparties (who need not know each other and who may be operating pseudonymously) who might otherwise use an oracle as depicted in Figure 1 can instead manually (and independently) observe an external piece of data upon which the on-chain distribution of funds is contingent, and can play a "game" to negotiate on-chain settlement without the need for an oracle at all (or as an alternative mechanism to backstop the potential technical failure of an oracle system). This situation is depicted schematically in Figure 3 below:

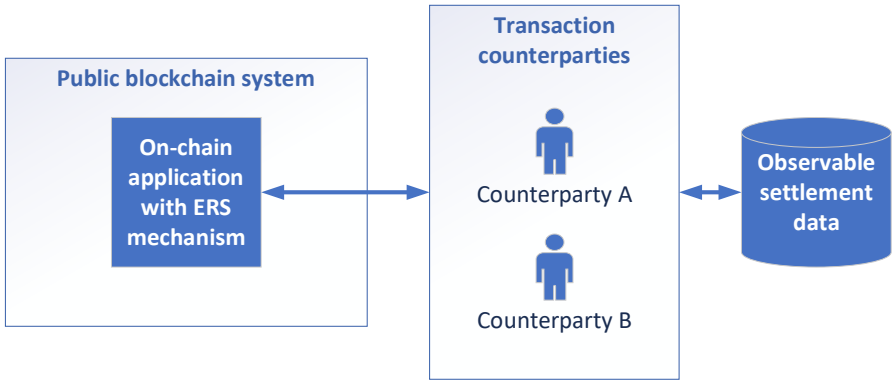

Figure 3: Schematic information flow through counterparties to a public blockchain, without an oracle.

Before describing how the ERS mechanism works, we first note that the settlement data must be observable by at least one counterparty, otherwise even an oracle-based system would not be able to

[15] Proprietary bets are another obvious class of financial agreement with similar technical characteristics as far as our proposal is concerned; we leave the distinction between financial derivatives and proprietary bets to the philosophers.

[16] An application-specific version of this concept (originally devised by one of the co-authors) is described in [2]; this paper generalizes the mechanism to a broader variety of transaction contexts.

reference the data in question (assuming that transactions are struck in a peer-to-peer fashion by counterparties interacting with an application on a public blockchain). In all likelihood, both counterparties to the underlying financial transaction must be able to observe the data, as if only one knew where the oracle would obtain the data, they would be unlikely to find a willing counterparty to the transaction at a mutually agreeable price due to issues of information asymmetry.[17] More fundamentally, two counterparties would only be likely to engage in a financial transaction contingent on some external data if that data were in some way relevant to them in the first place: there would be no rational reason to put funds at risk on a contingent outcome if the contingency itself were of no interest to one or both parties. Thus, we may safely assume that the settlement data are observable as depicted in Figure 3.

As one additional preliminary, we briefly describe the standard game of the prisoner's dilemma to illustrate models of adversarial behavior and to introduce the concept of a payoff matrix. The payoff matrix for the standard prisoner's dilemma game is illustrated in Figure 4 below:

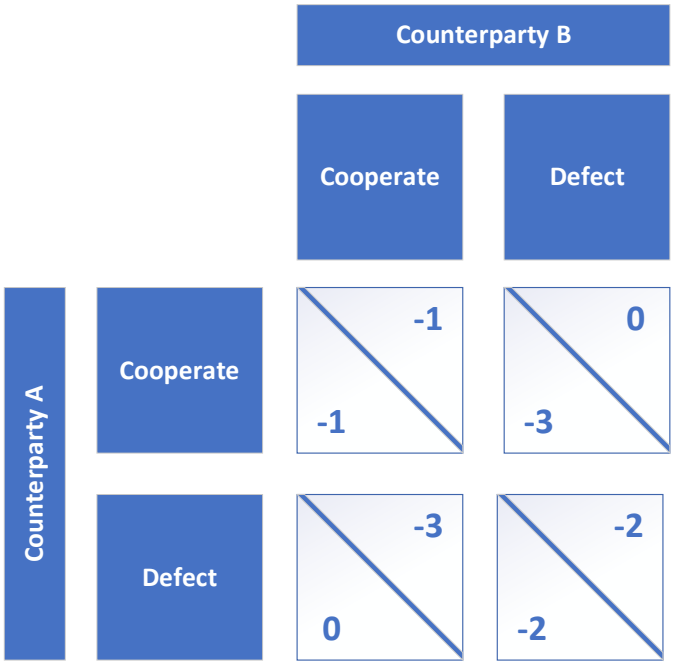

Figure 4: Payoff matrix for the standard prisoner's dilemma game.

Each axis in the payoff matrix represents one counterparty (in the standard game modeled as two prisoners who may defect by "ratting out the other" or cooperate by staying silent). Values above the diagonal line in each box of the payoff matrix represent the payoff to Counterparty B, and those below the line the payoff to Counterparty A. Negative values indicate a penalty (years of imprisonment, in the standard example); positive values would indicate positive utility to the counterparties. While we do not delve into the details of game theory here, we note that the dominant strategy for this game (that is, the economically rational decision for each counterparty to make, knowing the payoff matrix and being unable to negotiate with the other counterparty) is to defect, though it is easy to observe that the best mutual outcome is for both to cooperate.

[17] In practice, endpoints from which an oracle would fetch settlement data are likely to be stored as publicly visible data in the blockchain system itself.

With these concepts in mind, we now describe the (abstract) ERS mechanism itself. The mechanism must be applied in the context of an application where one or both counterparties to a transaction have put some economic value at stake, and the disbursement of that value is contingent upon external data that would otherwise be fetched by an oracle, as described above.[18] Put simply, the mechanism allows the two counterparties to a transaction to play a game wherein they can make one of two moves: offer a data point to be used for settlement (playing the role of oracle by proxy), or agree to the other party's offered data point.[19] By selecting an offered data point carefully, a counterparty may alter the payoff matrix for this game such that the dominant strategy is to cooperate (that is, for one counterparty to propose a reasonable data point for settlement, and for the other counterparty to accept it). The payoff matrix for the ERS game is depicted in Figure 5 below:

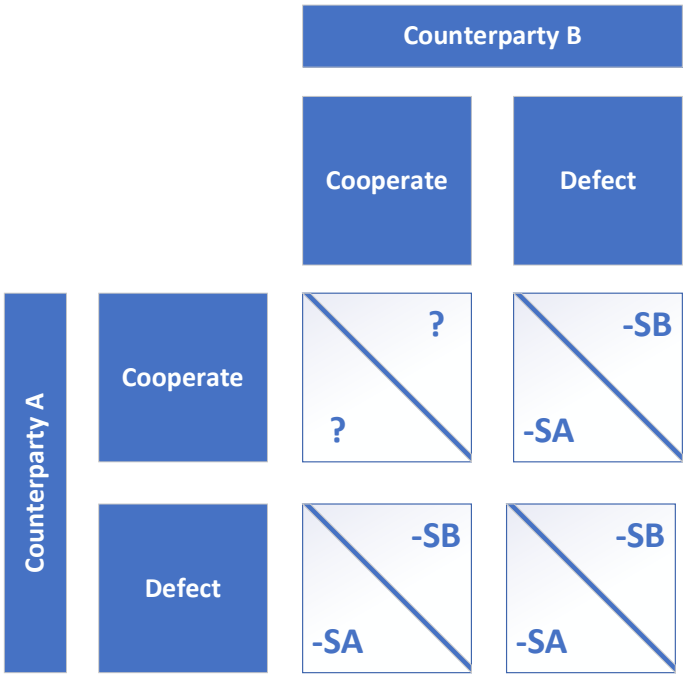

Figure 5: Stylized payoff matrix for the (iterated) ERS game.

This is an iterated game as both parties may make repeated offers (that the counterparty does not accept); however, in the context of public blockchains, transactions to record an offer on-chain are likely to incur fees, so there is an incentive for both parties to keep the number of iterations small. Fortunately, there is a Schelling point for the initial offer (made by either counterparty), which is the observable data that "would have" been used for settlement, had the system been using an oracle system

[18] While we used financial derivatives as an illustrative example of such an application, from another perspective, one might reasonably argue that any application meeting this description is by definition a financial derivative.

[19] The details of the implementation are in some sense "unimportant" for the general description of the mechanism, as they may differ depending on the exact application context, or could even be designed as a separate service application deployed to the same blockchain as the financial application wishing to use settlement services. What is crucial, however, is that the implementation of the ERS mechanism itself be secure in the context of the particular blockchain to which it is deployed. Depending on the target systems for deployment, these security concerns might vary, and are beyond the scope of the present paper. For any given target system, many known vulnerabilities are likely published, and implementations should take note of these to avoid the attack vectors they present.

directed to fetch the data from the known endpoint.[20] Before discussing behavior around this Schelling point, we consider the situations in which one or both counterparties defect by declining any offered data points. We indicate the financial interest that each counterparty has initially put at stake (and which will be subject to contingent disbursement by the on-chain application) by SA and SB for Counterparty A and Counterparty B, respectively. Note that either of those values might be zero (it would not be sensible for both of them to be zero, however).

Interestingly, in the case that either counterparty defects, both counterparties lose their entire stake by design of the game. This makes cooperation easier, as any value received by a counterparty should rationally be better than no value. However, the Schelling point value may itself produce a payoff to one counterparty that is either zero or very small, in which case the counterparty who would receive nothing by cooperating may elect to simply defect.[21] Economically rational counterparties should of course be aware of this situation, and can act accordingly. For situations in which one counterparty would receive de minimis value or possibly even incur a loss net of fees by cooperating, the other could offer an "adjusted" data point based on the actual observed external data point that favored the losing counterparty. With positive value at stake, it is always possible for the question marks in the cooperate / cooperate quadrant of the payoff matrix to be positive for both counterparties net of fees. If both counterparties are economically rational, one will propose a datapoint that causes these values to be sufficiently high to provide net positive utility despite any fees, while remaining suitably close to the original Schelling point value as to be considered "fair." Because oracles often take fees for their services, any such adjustment to incentivize cooperation in a counterparty could be considered an alternative form of such fee, and in the case that the Schelling point itself led to obvious net positive utility for both counterparties, rational actors should decide to cooperate directly on that value.

While this settlement framework may have some intuitive appeal compared to various oracle models discussed previously, it is not without its own vulnerabilities. Primarily, these arise in the case that at least one counterparty does not in fact behave economically rationally. For example, a counterparty may choose to "grief" the other, preferring for some reason to defect in order to force a penalty upon the other party rather than to cooperate and receive some positive reward. While such a griefing attack falls outside of the traditional model of economic rationality and thus is definitionally not handled by the ERS mechanism, this form of behavior remains a possibility in practice.[22] In essence, ERS trades off technical security vulnerabilities for economic security vulnerabilities. Which of these classes of vulnerabilities a pair of counterparties finds more acceptable is itself likely contingent upon a number of factors, such as the extent to which relevant software systems have been audited for security, expectations of the behavior of pseudonymous actors, past experiences transacting with different systems, and so forth. Fortunately, the choice of transaction settlement mechanisms is not mutually exclusive; just as some oracle systems take layered approaches to security, a blockchain application could opt to incorporate both an oracle layer and an ERS mechanism, with one serving as the fallback to the other.[23]

---

[20] A Schelling point is another concept from game theory describing a default solution that parties may select without communicating with one another, which is suitable for the case of a pseudonymous blockchain-based system.

[21] In the case that fees required for cooperative transactions outweighed the value that would accrue to a cooperating counterparty in such a scenario, defection would in fact be the economically rational decision.

[22] Reality is famously irrational at times, contrary to the theorizing of economists.

[23] In [2], a simplified version of this mechanism is proposed as a fallback for an oracle system, but these roles could be reversed, or both could be presented as options for counterparties to choose on a per-transaction basis.

Conclusion

Oracle systems for public blockchains attempt to solve an important problem for certain classes of on-chain applications that require data external to the blockchain itself, but the mechanisms for facilitating the transport of this data across the boundary of the blockchain system face a number of technical security challenges. These primarily fall under the categories of application vulnerabilities and network vulnerabilities, and these vulnerabilities exist at various degrees of separation from the application requesting off-chain data via an oracle service. Several approaches have been proposed and implemented that attempt to address oracle security from a technical perspective, or by introducing trusted third parties that can circumvent certain of these technical challenges. This paper proposes a novel generalized mechanism for broad classes of practical public blockchain applications dealing with the settlement of financial agreements that relies primarily on economic, rather than technical, security arguments. This economic security is vulnerable to its own classes of attacks (from what economists would describe as irrational actors, rather than threat actors in the parlance of computer security); thus, in practice, it may best be deployed as an additional part of a layered system designed to carry off-chain data to an application resident on a public blockchain.

References

[1] Antonopoulos and Wood, 2018. “Mastering Ethereum.” O’Reilly Media, Inc.

[2] Arief, Algya, and Lee, 2019. “SmartPiggies: An open-source standard for a free peer-to-peer global derivatives market.” Available: https://www.smartpiggies.com/.

[3] Zhang, Cecchetti, Croman et al., 2016. “Town Crier: An Authenticated Data Feed for Smart Contracts.” Available: https://eprint.iacr.org/2016/168.pdf.

[4] Chainlink: https://chain.link/.

[5] Provable (formerly Oraclize): https://provable.xyz/.

[6] Nilsson, Bideh, and Brorsson, 2020. “A Survey of Published Attacks on Intel SGX.” Available: https://arxiv.org/pdf/2006.13598v1.pdf.

[7] UMA Oracle Service: https://docs.umaproject.org/getting-started/oracle.

[8] Kleros: https://kleros.io/.

[9] MakerDao: https://makerdao.com/en/.