UCSB CS 291D: Blockchains and Cryptocurrencies **Fall 2020**

Lecture 9: Connecting Blockchains with Real World Using Oracles

Lecturer: Fan Zhang *Scribes: Priyanka Bose, Fatih Kaleoglu, Lawrence Lim*

9.1 The Oracle problem

Smart contracts are programs executed by blockchain nodes. Smart contracts are general purpose programs that provides integrity and availability. Almost anything can be programmable using a smart contracts and can be programmed by high-level languages such as Solidity,and Vyper. Integrity implies that their execution can not be stopped i.e, they can not be tampered with. Availability means that it is very hard to stop smart contract from running.

Though smart contracts are useful, an important barrier for its widespread adaptability lies in its inability to fetch real-world data as smart contracts can not access external data. They can only read from on-chain data. To be able to access external data, smart contracts use oracles. The concept of an oracle is very simple. They behave as an intermediary, i.e. they are be able to fetch data from the external sources and push data back to the block-chain. The oracle should provide strong security and privacy guarantees. Specifically, the oracle should have the following properties:

- Integrity: Oracle should not be misreporting
- Privacy: Oracle should be able to access relevant data of a user without getting to know any private data related to the user
- Legacy compatibility: It should be compatible with all external websites.

9.1.1 Challenges

Legacy Compatibility. Various websites serve external data. Therefore, the oracle should be compatible with all such existing websites.

Integrity. Integrity can be boosted by decentralization i.e., by using multiple oracles. The idea is to take majority vote of the oracles. Therefore, if one of the oracle is corrupt, it is still fine. In real world, we have decentralized oracle network consisting of several nodes run by different organizations. The assumption is k out of n nodes are honest.

Privacy. Privacy is one of the hardest problem to solve. Specifically, an oracle should be able to extract required information from an user without knowing any private data related to the user. Let's say an oracle needs to verify whether an user Alice has greater than 5000 bank balance. Therefor Alice should be able to convince the oracle it is indeed the case without revealing any more details related to her bank account such as password etc. That means, Alice (Prover) need to prove to the oracle (Verifier) that she has a balance of > 5000 in her account. Alice's proof should satisfy the following properties:

- Legacy compatibility
- Alice can not cheat

- Verifier can not know Alice's password

One possible way to do this is could be to use TLS. However, TLS does not sign data. Therefore, there is no proof that Alice is not cheating. To do this using TLS one needs to enable TLS signing capabilities, however for that lot of websites depending on TLS need to be changed, which is not feasible since it violates the property of legacy compatibility. DECO attempts to solve this problem

9.2 DECO (Decentralized Oracle)

The main idea behind DECO is to split the MAC key between Alice (P) and Verifier (V) so that Alice cannot cheat. The DECO protocol can be summarized as follows:

1. Three-party handshake
2. TLS as usual (MPC between V-P)
3. P either decrypts or proves in ZK to V.

9.2.1 Three-Party Handshake

Let's first consider the two-party case:

Diffie-Hellman Key Exchange (Two parties) Let[1] $\mathcal{G}$ be a cyclic group with generator g and assume that discrete-logarithm[2] is hard in $\mathcal{G}$. Both the server and the client own secret keys x_s, x_k, respectively. To agree on a common key, the parties send each other (public) keys $y_s = g^{x_s}$ and $y_c = g^{x_c}$ over a public channel. The shared key is $z = g^{x_s x_c} = y_s^{x_c} = y_c^{x_s}$, which can be computed by both the server and the client.

Three-Party Key Exchange In the three-party case, the prover (Alice) and the verifier act collectively as the client for Diffie-Hellman. As before, the three parties own their secret keys denoted by (x_s, x_p, x_v). For the public key, Alice sends to the server $y_c = y_p \cdot y_v = g^{x_p} \cdot g^{x_v}$. The shared key then is given by

$$z = y_c^{x_s} = g^{x_s(x_p+x_v)} = g^{x_s x_p} \cdot g^{x_s x_v} = y_s^{x_p} \cdot y_s^{x_v} = z_p \cdot z_v$$

.

Therefore, Alice and the verifier own parts of the key z_p and z_v which multiply up to the server's key z.

9.2.2 Running the TLS protocol

The encryption and MAC keys used in TLS are derived using a pseudo-random function, i.e. $(K^{\text{Enc}}, K^{\text{MAC}}) = \text{PRF}(z) = \text{PRF}(z_p \cdot z_v)$. The TLS server can compute the keys directly, yet Alice needs to engage in secure multiparty computation with the verifier in order to compute the keys while z_p and z_v remain private. This results in Alice and the verifier sharing the MAC key K^{MAC}, holding respectively K_P^{MAC} and K_V^{MAC} such that $K^{MAC} = K_P^{MAC} \bigoplus K_V^{MAC}$.

[1] In DECO, the elliptic key version of this is used (ECDHE).

[2] In reality the Diffie-Hellman assumption is slightly weaker than that of discrete-log, only assuming the hardness of recovering (x_s, x_c) given $(g^{x_s}, g^{x_c}, g^{x_s x_c})$.

After that, Alice interacts with the TLS as usual, resorting to multiparty computation with V as needed. It is not until Alice has sent the server response to the verifier that the verifier reveals to Alice its portion of the key K_V^{MAC}. Because TLS ciphertexts are cryptographical commitments, this prevents Alice from cheating.

A note on efficient multiparty computation: z_P and z_V are points on the elliptic curve used in the key exchange, and computation in this domain can be very costly. One of the main ways to overcome this barrier is share conversion utilizing Paillier cryptosystem. Using this method, group addition can be carried out in the underlying prime field instead of the elliptic curve itself.

9.2.3

In the final step, Alice generates a proof. She can utilize zero-knowledge proofs to control the amount of information shared with the verifier, e.g. she can prove having funds greater than 5000 dollars in her bank account without revealing any other information about the exact amount she has.

After Alice convinces multiple oracle nodes of her statement, they can issue a certificate on her behalf to be used in a smart contract.

9.2.4 Applications

Oracles and the DECO protocol provide several applications to smart contracts:

9.2.4.1 Confidential DeFi

One possible application of oracles is in decentralized finance. A centralized finance system has one trusted body that handles finance transactions, whereas a decentralized finance system has permissionless users with no central authority managing transactions. One use of oracles is to create a smart contract so that if an event is true, then give the money to one user Alice, or otherwise give it to the other user Bob. Both Alice and Bob would commit their money to the smart contract, which would then query a body of oracles the help decide whether the event is true or false, and with the oracles' responses, award the money to the correct user.

One improvement is making the event confidential to the smart contract and oracles. In order to hide the event, Alice can use DECO to interact with a trusted third-party and oracles with a zero-knowledge proof to convince the oracles that Alice should win the money in the smart contract. Because of the zero-knowledge proof, the oracles also learn nothing about the event and can then tell who won the money to the smart contract, which also learns nothing about the event except who won.

9.2.4.2 Decentralized Identity

Another important use case of DECO is for decentralized identity. Proving identity is an important application as granting access to resources or other privileges is useful in many cases where needing to trust a user's properties is important. With the three party handshake in the DECO protocol, oracles can interact with a trusted third-party such as a bank, the government, or other trusted agency and certify the identity of a user (such as the birthdate). With a zero-knowledge proof, the DECO protocol can even hide all information

except what needs to be proved. (For instance, if Alice needs to prove that she is 18 or older, then a zero knowledge proof can hide her birthday while still proving that she is 18 or older.)

9.2.5 Limitation of DECO

One major limitation of DECO is that the interactive protocol requires both a prover and a verifier. This means that an intermediary like an oracle is required for this protocol, so a prover cannot commit to the blockchain without proving to a verifier first. However, it has not been shown to be impossible to prove to the blockchain without an oracle, so there is an open problem there.

9.3 Other Oracle Approaches

- **Trusted Hardware** - Town Crier (CCS'16)

  Trusted hardware is hardware that has been designed and certified to perform certain operations. It is an effective solution that solves both integrity and confidentiality but relies on the trust of this hardware to guarantee security.

- **TLS modification** - TLS-N (NDSS'19)

  Another good option is modifying the TLS handshake to make these oracle operations easier. A server or trusted third-party can sign their data so that authenticity is guaranteed. One problem with this solution is that adoption can be difficult when changing a widely adopted security standard. DECO is adapted to work with the current TLS handshake using MPC.

- **Game Theoretic Constructions** - ShellingCoin (Buterin, 2014), Astrea: A Decentralized Blockchain Oracle (Alder et al, 2018)

  A game theoretic construction aims to make revealing the truth the Nash Equilibrium. In other words, revealing the truth is the best possible outcome for parties involved. These solutions are interesting in that they can be combined with other oracle solutions.

- **On-chain data as oracles** - Uniswap

  The blockchain itself stores data and this data can be used as an oracle to power other blockchain applications. One example application is an oracle for blockchain price: Because many exchanges are happening on the blockchain, the price of a cryptocurrency can be learned from data on the blockchain. These on-chain price feeds would be authenticated, but are specific to asset prices and subject to manipulation, even attacks!

References

[1] Fan Zhang, Sai Krishna Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. Deco: Liberating web data using decentralized oracles for tls. *CoRR*, abs/1909.00938, 2019.