How to Make a Prediction Market on Twitter with Bitcoin

Lloyd Fournier

lloyd.fourn@gmail.com

Abstract. In his work *Discreet Log Contracts* (DLC) Dryja presents a compelling design for smart contracts that settle based on real world events. We show how to optimise the construction for events with binary outcomes. Our protocol requires only two rounds of communication for the contract setup and only two on-chain transactions. The setup messages are small enough so that they can be encoded in a single tweet or text message. This circumvents the need for a prediction market exchange and peer-to-peer communication channel. Instead, users use their existing social media and messaging channels to both find other users to bet with and to send the protocol messages.

1 Introduction

In the blockchain world, an "Oracle" is a trusted party who attests to the state of the real world so that *smart contracts* may settle based on real world events. There are a variety of proposals for how oracles can perform this function. The most compelling model for Bitcoin-like systems is described the Dryja's *Discreet Log Contracts* (DLC)[4]. In this model, oracles do not interact with the blockchain nor do they even necessarily interact with the parties involved in the contract. Instead an oracle makes a commitment per possible outcome and then, once the event has transpired and the outcome is known, the oracle opens the commitment corresponding to that outcome. Concretely, the oracle commits to a group element per outcome and reveals the discrete logarithm. The original scheme used a Schnorr-like construction to specify a group element per outcome with a constant size message. For our protocol, we are only concerned with binary outcome events so it suffices to assume for each event the oracle specifies two outcome group elements (O_0, O_1) per event.

The most attractive aspect of the DLC model is that the oracle is an *oblivious trusted third party*[3]. Despite being trusted, they are kept oblivious to any concrete instance the protocol using them as an oracle (unless they are party to one). This is true even if the oracle is monitoring the blockchain. The oracle (or anyone else) should be unable to distinguish the transactions making a bet on event A using oracle B from transactions betting on event C using oracle D. This is in stark contrast to prediction markets based on Ethereum[6] where users post their bets publicly on-chain and the oracles interact directly with the smart contract on-chain.

1.1 Our Contribution

We propose a novel solution to two open questions surrounding Discreet Log Contracts: (i) how should users discover other users to bet against? and (ii) how should they communicate? Our idea is to leverage existing communication channels that users are already comfortable with such as social media and text messaging applications. Through these, users simulataneously find other users to bet against and communicate the necessary cryptographic information to complete the contract. Towards this end, we have optimised the original protocol to only two rounds of communication, where one party *proposes* an event to bet on and then others post offers in reply. Both messages fit into a single 240 character tweet (assuming a maximum of 385 bytes per tweet [7]).

Our protocol is designed for the public message board setting which is naturally less private than assuming a direct communication channel. Nevertheless, by embedding a Diffie-Hellman key exchange in the messages we manage to retain some important privacy properties even in this setting.

2 The Protocol

2.1 Overview

Like a typical Discreet Log Contract our protocol allows two parties to make a bet on the outcome of an event. Our protocol only works with events with binary outcomes and it is winner takes all. Neither party is assumed to have communicated with each other beforehand or have a direct communication channel. They are both expected to have an incorruptible view of the blockchain and to be able to read from and write to a public message board e.g. Twitter. Additionally both parties must be able to access the oracle's commitments and attestations.

The protocol begins with Alice posting a *proposal* to the public message board. The proposal includes the event she wants to bet on and how much she is willing to risk (not how much she wants to gain). In response, parties we will name Bob, post offers. Bob's offer contains how much he is willing to bet on a particular outcome happening and signatures on inputs providing that amount to a Fund transaction. Crucially, the proposal and offer contain public keys which are used in the `OP_CHECKMULTISIG` output of the Fund transaction and to create a Diffie-Hellman shared secret so Bob's offer can be encrypted. This completes the setup. Now the parties wait to learn the discrete logarithm of one of the outcome keys, after which one of them will be able to claim the coins. Note that we do not consider the possibility that the oracle could be malicious and assume they release the discrete logarithm of the correct key in a timely manner.

2.2 Notation

We denote Bitcoin's Secp256k1 elliptic curve group as $\mathbb{G}$ its order as q, its fixed generator as G and its group operation as addition. Each party has access to a hash function for key derivation $H_{\mathsf{KDF}} : \mathbb{G} \to \{0,1\}^l$ where $l \geq 128$ is a

reasonable choice for Bitcoin and a pseudorandom generator $\mathsf{PRG} : \{0,1\}^l \to (\{0,1\} \times \mathbb{Z}_q^2 \times \{0,1\}^{|m|})$ where $|m|$ is the length of the payload of an offer. Both parties are assumed to have a Bitcoin wallet offering functionality (Value, $\mathsf{Witness}$, $\mathsf{VerifyWitness}$) which (i) return the total value of a list of inputs, (ii) generate a witness for a given set of inputs in transaction and (iii) verify the validity of the given transaction witness.

2.3 Alice's Proposal

To make her proposal, Alice decides the amount of coins β_0 she wants to bet on a particular event identified by $\mathsf{event\_id}$. She provides a list I_0 of n_0 inputs which hold at lest β_0 Bitcoin in total and an (optional) change address change_0 to receive the difference. Finally, she chooses a private key $x_0 \leftarrow_\$ \mathbb{Z}_q$ and computes her corresponding public key $X_0 \leftarrow g^{x_0}$. She posts these to the message board. Each component and its size is shown in Figure 1.

Symbol	Description	Size (bytes)
$\mathsf{event\_id}$	A descriptor of the oracle and event the bet is on	id
β_0	The amount of coins that Alice will bet	6
I_0	The inputs that Alice is using to fund the bet	$33n_0$
change_0	A change address	20
X_0	Alice's public key	32
	Total	$58 + 33n_0 + \mathsf{id}$

Fig. 1. Alice's proposal where n_0 is the number of inputs she uses to fund.

An honest Alice who has received offers, but no desirable offer before the event outcome time should cancel her proposal by spending one of the inputs in I_0. We discuss how to deal with a malicious Alice does not do this in Section 3.2.

2.4 Bob's Offer

Anyone who wishes to make an offer in response to Alice's proposal we will name Bob. Bob chooses which outcome $c \in \{0,1\}$ he wants to bet on and how much he wants to bet on it β_1. Each component of the offer and its size is described in Figure 2. The ratio of β_0 to β_1 determine the odds for the bet as the winner will take $\beta_0 + \beta_1$. In Bitcoin as it is today, the size of Bob's offer is $63 + 130n_1$ bytes where n_1 is the number of inputs he uses to fund the bet. Each witness contains an ECDSA signature and the input's public key. With base2048 encoding[7] we have a limit of 385 bytes per tweet so Bob can have at most two inputs.

Bob generates his offer using the Offer algorithm in Figure 3 given Alice's proposal P_0. First he computes the Diffie-Hellman shared secret X and uses the output of PRG to encrypt the payload of the message and to randomize the

public keys that appear in the on-chain contract (including the order in which they appear). Without randomisation, it would be easy to link the public keys in the 1-of-2 `OP_CHECKMULTISIG`[2] (denoted $\texttt{OP\_CMS}_{\text{1-of-2}}$) output to the keys in the messages.

Note that Bob can cancel his offer by successfully spending any of the inputs in I_1 (which should be easy if Alice has not confirmed the Fund transaction yet). If Alice takes a different offer, then this notionally cancels his offer as well because it also invalidates the Fund transaction.

Symbol	Description	Size (bytes)
X_1	Bob's public key	32
	The following are encrypted as the ciphtertext e	
c	The outcome $c \in \{0,1\}$ that Bob wants to bet on	1
I_1	The input(s) that Bob is using to fund the bet	$33n_1$
σ_1	The witness for each input	$97n_1$
β_1	The value that Bob will wager	6
fee	The fee that will be paid by Bob	4
change_1	A change address	20
	Total	$63 + 130n_1$

Fig. 2. Bob's offer in response to Alice's proposal. His public key X_1 is visible while the payload is encrypted using the Diffie-Hellman shared secret between X_0 and X_1.

$\mathsf{Offer}(P_0, (I_1, \mathsf{change}_1, \beta_1, \mathsf{fee}))$	$\mathsf{GenFund}(P_0, P_1, (b, r_0, r_1))$
$x_1 \leftarrow_\$ \mathbb{Z}_q; X_1 \leftarrow x_1 G;$	$(X_0, I_0, \mathsf{change}_0, \beta_0) := P_0$
$P_1 := (X_1, I_1, pk_1, \mathsf{change}_1, \beta_1, \mathsf{fee})$	$(X_1, I_1, \mathsf{change}_1, \beta_1, \mathsf{fee}) := P_1$
$X \leftarrow X_0^{x_1}; sk \leftarrow H_{\mathsf{KDF}}(X)$	$Z_0 \leftarrow X_0 + r_0 G + O_{1-c}$
$(b, r_0, r_1, k) \leftarrow \mathsf{PRG}(sk)$	$Z_1 \leftarrow X_1 + r_1 G + O_c$
$\mathsf{Fund} \leftarrow \mathsf{GenFund}(P_0, P_1, (b, r_1, r_0))$	$\mathsf{Fund} \leftarrow \mathsf{Tx}($
$\sigma_1 \leftarrow_\$ \mathsf{Witness}(I_1, \mathsf{Fund})$	`inputs:` $I_0 \| \| I_1$,
$m \leftarrow I_1 \| \sigma_1 \| \beta_1 \| \mathsf{fee} \| \mathsf{change}_1$	`outputs:` [
$e \leftarrow k \oplus m$	$(\texttt{OP\_CMS}_{\text{1-of-2}}(Z_b, Z_{1-b})), \beta_0 + \beta_1),$
return $(e, (x_1, X_1))$	$(\mathsf{change}_0, \mathsf{Value}(I_0) - \beta_0),$
	$(\mathsf{change}_1, \mathsf{Value}(I_1) - \beta_1 - \mathsf{fee})$
	])
	return Fund

Fig. 3. The Offer algorithm Bob uses to generate his encrypted offer. The deterministic $\mathsf{GenFund}$ algorithm is used by both Alice and Bob to generate the Fund transaction. The oracle will reveal the discrete logarithm of O_0 or O_1.

2.5 Alice Takes Offer

Alice decrypts the encrypted payload e from each offer (X_1, e) she receives to check if it is desirable and to verify its contents. If she wishes to accept the offer she signs her set of inputs I_0 with her wallet and broadcasts the Fund transaction as depicted in Figure 4.

$\mathsf{TakeOffer}((P_0, x_0), (X_1, e))$

$X \leftarrow X_1^{x_0}; sk \leftarrow H_{\mathsf{KDF}}(X)$
$(b, r_0, r_1, k) \leftarrow \mathsf{PRG}(sk)$
$(I_1, \sigma_1, \beta_1, \mathsf{fee}, \mathsf{change}_1) \leftarrow k \oplus e$
$P_1 := (X_1, I_1, \beta_1, \mathsf{fee}, \mathsf{change}_1)$
$\mathsf{Fund} \leftarrow \mathsf{GenFund}(P_0, P_1, (b, r_0, r_1))$
$\mathsf{VerifyWitness}(I_1, \sigma_1, \mathsf{Fund}) \stackrel{?}{=} 1$
$\sigma_0 \leftarrow\!\!\$\ \mathsf{Witness}(I_0, \mathsf{Fund})$
return $((I_0, \sigma_0), (I_1, \sigma_1), \mathsf{Fund})$

Fig. 4. Alice takes the encrypted offer (X_1, e) by completing the witnesses needed for the Fund transaction and broadcasting it

2.6 Oracle Attests to Outcome

When the real world event has transpired and the oracle has observed its outcome $w \in \{0, 1\}$ they publish the discrete logarithm o_w of the corresponding outcome key O_w. Upon seeing o_w, the ith party who bet on outcome w can compute the private key for Z_i as $z_i \leftarrow x_i + b_i + o_w$. With z_i they can generate a ECDSA signature under Z_i and therefore spend output of the 1-of-2 `OP_CHECKMULTISIG` output of the Fund transaction in a transaction of their choosing.

3 Security

We now briefly provide an intuition for the security of the protocol. First and foremost we must guarantee that only the party that wins can claim the funds. The original DLC paper[4] did not provide a formal security proof and unfortunately seems to be vulnerable to *rogue key* attacks often found in naive multisignature schemes[5]. The issue is that either party can maliciously generate their public key so that it cancels out one of one of the outcome keys. For example, in our scheme if Alice chooses $X_0 = aG - O_{1-c}$ for some $a \leftarrow\!\!\$\ \mathbb{Z}_q$ then the private key of $Z_0 = X_0 + r_0 G + O_{1-c}$ will be $a + r_0$ which Alice can easily compute and thereby steal the funds before the outcome is even known. Fortunately, there are

multiple way to address this issue. In the case of our protocol, we have somewhat unwittingly made this attack implausible under the *Knowledge of Exponent Assumption* (KEA1) [1] by making the two parties compute the Diffie-Hellman shared secret X between their two keys.

We sketch the proof of security against a malicious Alice who steals the bet funds by creating a ECDSA signature under Z_0 without the oracle revealing the discrete logarithm of O_{1-c}. Alice must query H_{KDF} with X which in the random oracle model presents it to the proof environment. Under KEA1 we can extract x_0 from (X_0, X). If we assume that the Discrtete Logarithm Problem in $\mathbb{G}$ can be reduced to forging ECDSA signatures, we can extract z_0 from the signature. Thus we can compute $o_{1-c} \leftarrow z_0 - x_0 - r_0$ which contradicts our premise that Alice was able to take the coins without knowing o_{1-c}.

We do not offer a full proof of security in this short paper but we wish to note that: (i) the proof for Bob spending with a signature under Z_1 without knowing the discrete logarithm of O_c is practically the same and (ii) an Alice who spends with a signature under Z_1 (or Bob under Z_0) can clearly forge ECDSA signatures.

3.1 Privacy

Out of necessity Alice posts her initial bet proposal, with her inputs and change address, on a public message board to attract offers. This is clearly a loss of privacy for Alice as it associates these with her identity on the message board e.g. Twitter username. This also allows passive observers to link the inputs Alice publicly chooses to the on-chain Fund transaction that spends them.

Despite this, our protocol ensures two core privacy properties against a passive observer who has seen the protocol messages, the on-chain transactions and the oracle's attestation: (i) *Winner Privacy*, the observer should not have any advantage in guessing whether Alice or Bob won the bet (ii) *Offer Privacy*, the observer should not be able to distinguish the real offer that Alice actually accepted from those that she didn't accept. We will not prove these properties in this short paper, but conjecture that any advantage an observer would have against these would violate the assumption that PRG is a pseudorandom generator or that the Computational Diffie-Hellman problem is hard.

3.2 Post-Event Acceptance Attack

There is nothing to stop a malicious Alice from accepting an offer after the outcome is already known. In our protocol, it is Bob's responsibility to protect himself against this by cancelling his offer well before the outcome is known. This is undesirable as a malicious Alice could attract hundreds of offers and if any of them is not cancelled in time she has a risk-free chance of taking their coins. Additionally it would require hundreds of transactions to successfully cancel all the offers. Thus, it may be prudent to only make offers to parties who have a reputation of being prompt in accepting offers or cancelling their proposal on-time.

Using Bitcoin's time-lock feature we can construct a less costly solution. Unfortunately, Bitcoin does not have the notion of a transaction that is *valid* only until a certain time. This would immediately solve our problem. Bitcoin does offer the ability to make a transaction *invalid* until a certain time. Consider the following modification to the protocol: in the proposal message Alice additionally provides a signature on a time-locked transaction spending one of her inputs. The time-lock would be set to expire before the outcome is expected to be known. Anyone who sees the proposal could then broadcast this transaction after the expiry time (if she has not accepted an offer or cancelled it herself), thereby cancelling the proposal and all offers based on it in one transaction. The transaction could even spend the input to a burn address (where the funds would be unrecoverable) to incentivise Alice to be proactive about cancelling her proposal early.

References

1. Bellare, M., Palacio, A.: The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. Cryptology ePrint Archive, Report 2004/008 (2004), `https://eprint.iacr.org/2004/008`
2. Bitcoin Wiki: Script, `https://en.bitcoin.it/wiki/Script`, accessed on 2020-01-10
3. Camenisch, J., Haralambiev, K., Kohlweiss, M., Lapon, J., Naessens, V.: Structure preserving cca secure encryption and its application to oblivious third parties. Cryptology ePrint Archive, Report 2011/319 (2011), `https://eprint.iacr.org/2011/319`
4. Dryja, T.: Discreet log contracts. `https://adiabat.github.io/dlc.pdf` (2017)
5. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Cryptology ePrint Archive, Report 2018/068 (2018), `https://eprint.iacr.org/2018/068`
6. Peterson, J., Krug, J.: Augur: a decentralized, open-source platform for prediction markets. CoRR **abs/1501.01042** (2015), `http://arxiv.org/abs/1501.01042`
7. qntm: base2048, `https://github.com/qntm/base2048`, accessed on 2020-01-10