

Security of Discreet Log Contract Attestation Schemes

Lloyd Fournier

lloyd.fourn@gmail.com

1 Introduction

In *Discreet Log Contracts*[2], Dryja presents a compelling design for Bitcoin based smart contracts that settle based on real world events. The real world outcome of an event is mapped into Bitcoin transactions through entities called *oracles* who provide a cryptographic attestation to a particular outcome.

In this short work we provide security definitions and proofs that were absent from the original work. Furthermore, we remark that the scheme can be simplified and achieve the same security properties without assuming a collision resistant hash function. This work is unfinished and so far only covers the notion of attestation unforgeability.

Definition 1 (Oracle Attestation Scheme). *An oracle attestation scheme consists of four efficient algorithms:*

- $\text{KeyGen}(\cdot) \mapsto (\text{sk}, \text{pk})$: *A key generation algorithm*
- $\text{Announce}(\text{sk}, \text{event_id}) \mapsto (r, a)$: *An event announcement algorithm which the oracle uses to generate an announcement secret r and an announcement a .*
- $\text{Attest}(\text{sk}, \text{event_id}, r, a, o) \rightarrow t$: *An event outcome attestation algorithm which the oracle uses to publicly attest to the fact that an event identified previously announced with a has the outcome o .*
- $\text{Verify}(\text{pk}, \text{event_id}, a, t, o) \rightarrow \{0, 1\}$: *An attestation verification algorithm which verifies that an attestation t attests to o being the outcome of an event with announcement a under the oracle's public key pk .*

It's important to note that the above definition does not provide an intuition as to why an oracle attestation scheme is useful. Users have to be able to use the announcement a to anticipate some form of the final attestation for each particular outcome and use it to lock funds on the ledger such that the revelation of the corresponding attestation t will have the desired effect. This is why **Attest** is always deterministic as it simply reveals what was previously committed to in the announcement. We do not capture this crucial idea in the definitions since it is not important for defining security for the oracle. Readers should also note we intentionally leave the structure of **event_id** to your imagination. The only constraint we have on it is that it uniquely identifies a real world event and all possible valid outcomes of the event.

2 Security Definitions

2.1 Attestation Unforgeability

Attestation unforgeability refers to the notion that if a valid attestation exists with respect to a particular announcement under an oracle's public key then it must have been created by that oracle. An attestation is only valid with respect to a particular *announcement*. Make sure not to confuse the notion of *oracle* in the sense of a DLCs with the **Att** and **Ann** oracles defined below which are *oracles* in the sense of a security game definition.

Definition 2 (Attestation Unforgeability). *An algorithm $\mathcal{F}(\tau, \epsilon, n)$ -breaks the unforgeability of an oracle attestation scheme (**KeyGen**, **Announce**, **Attest**, **Verify**) if runs in at most time τ makes at most n queries to the **Ann** and **Att** oracles and $\Pr[\text{Forge} = 1] \leq \epsilon$ in the experiment defined below.*

Forge	Oracle Ann (event_id _i)
$Q_{\text{Ann}} := \emptyset; Q_{\text{Att}} := \emptyset$	$(r_i, a_i) \leftarrow \text{Announce}(\text{sk}, \text{event_id}_i)$
$(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}$	$Q_{\text{Ann}} := Q_{\text{Ann}} \cup \{(\text{event_id}_i, r_i, a_i)\}$
$(\text{event_id}^*, t^*, o^*) \leftarrow \mathcal{F}^{\text{Ann, Att}}(\text{pk})$	return a_i
return	
$\exists(\text{event_id}^*, \cdot, n^*) \in Q_{\text{Ann}} \wedge$	
$\exists(\text{event_id}^*, o_j) \in Q_{\text{Att}} \wedge$	
$o^* \neq o_j \wedge$	
$\text{Verify}(\text{event_id}^*, n^*, t^*, o^*)$	
	<hr/>
	Oracle Att (event_id _i , o _i)
	check $\exists(\text{event_id}_i, r_i, n_i) \in Q_{\text{Ann}} \wedge (\text{event_id}_i, \cdot) \notin Q_{\text{Att}}$
	$t_i \leftarrow \text{Attest}(\text{sk}, \text{event_id}_i, r_i, n_i, o_i)$
	$Q_{\text{Att}} := Q_{\text{Att}} \cup \{(\text{event_id}_i, o_i)\}$
	return t_i

Without loss of generality we assume that \mathcal{F} always queries **Att** with an **event_id** if it has queried **Ann** with it before returning its forgery. However it may do so in any order. We also assume that \mathcal{F} only queries valid outcomes o to **Att** with respect to the **event_id**.

In Figure 1 we present our interpretation of Dryja's original scheme framed according to Definition 2. The scheme resembles a deconstructed version of the Schnorr signature scheme where the nonce R is pre-committed to during the announcement and then the s value of the signature is revealed during the attestation. We then go on to show that the scheme is secure if the *one-more discrete logarithm problem* is hard by reduction.

KeyGen	Announce(sk, event_id)
$x \leftarrow \mathbb{Z}_q$	$r \leftarrow \mathbb{Z}_q$
$X \leftarrow xG$	$R \leftarrow rG$
$\text{sk} := (x, X); \text{pk} := (X);$ return (sk, pk)	return (r, R)
Attest(sk, event_id, r, a, o)	Verify(pk, event_id, a, t, o)
$(x, X) := \text{sk}; R := n$	$R := n; s := t; X := \text{pk}$
$c \leftarrow H(R, X, \text{event_id}, o)$	$c \leftarrow H(R, X, \text{event_id}, o)$
$s \leftarrow r + cx$	return $R = sG - cX$
return s	

Fig. 1. Dryja's Schnorr-like oracle attestation scheme.

Definition 3 (One-More Discrete Logarithm Problem [1]). Let \mathbb{G} be a cyclic group of order q and G be a generator of \mathbb{G} and let $D\text{Log}_G$ be an oracle which returns the discrete logarithm of the queried element with respect to G . An algorithm \mathcal{A} is said to (τ, ε, n) -solve the OMDL problem in \mathbb{G} if it runs in at most time τ , makes at most n queries to $D\text{Log}_G$ and $\Pr[\text{OMDL} = 1] \geq \varepsilon$ for the OMDL experiment below.

OMDL
$x_0, \dots, x_n \leftarrow \mathbb{Z}_q^{n+1}$
for $i = 0, \dots, n$ do
$X_i \leftarrow x_i G$
$x_0^*, \dots, x_n^* \leftarrow \mathcal{A}^{D\text{Log}_G}(X_0, \dots, X_n)$
return $(x_0^*, \dots, x_n^*) = (x_0, \dots, x_n)$

Theorem 1 (OMDL implies Attestation Unforgeability). Let H be the hash function that that Dryja's scheme is instantiated with. Let \mathcal{F} be a forger that (τ, ε, n) -breaks the attestation unforgeability of Dryja's scheme. Then there exists an adversary that (τ', ε', n) -solves the OMDL problem and an adversary that $(\tau_{\text{cr}}, \varepsilon_{\text{cr}})$ -breaks the collision resistance of H such that

$$\varepsilon \leq \varepsilon' + \varepsilon_{\text{cr}}, \tau' = \tau + O(n), \tau_{\text{cr}} = \tau + O(n)$$

Proof. We use a forger \mathcal{F} against the unforgeability of Dryja's scheme to construct an adversary \mathcal{A} against OMDL. We take the first challenge group element X_0 and use it as our oracle's public key. Our strategy is to then use the $D\text{Log}_G$ oracle to simulate the attestation oracle Att after first using the X_1, \dots, X_n as

the announcements. Once \mathcal{F} returns a forged attestation we use it to extract the discrete logarithm of X_0 and work backwards to find the discrete logarithm of the other challenge elements.

$\mathcal{A}^{\text{DLog}_G}(X_0, \dots, X_n)$	Simulate Ann(event_id _{<i>i</i>})
1: $(i^*, t^*, o^*) \leftarrow \mathcal{F}^{\text{Ann, Att}}(X_0)$	$a := X_i$
2: $(\text{event_id}^*, t, c, o) := R_{i^*}$	$i = i + 1$
3: $c^* \leftarrow H(X_{i^*}, X_0, \text{event_id}^*, o^*)$	return a
4: if $c = c^*$ then abort	
5: $x_0 \leftarrow (t - t^*) / (c - c^*)$	Simulate Att(event_id _{<i>i</i>} , o_i)
6: for $i = 1, \dots, n$ do	$c_i \leftarrow H(X_i, X_0, \text{event_id}_i, o_i)$
7: $(\text{event_id}_i, t_i, c_i, o_i) := R_i$	$S_i \leftarrow X_i + cX_0$
8: $x_i \leftarrow t_i - c_i x_0$	$t_i \leftarrow \text{DLog}_G(S_i)$
9: return (x_0, \dots, x_n)	$R_i := (\text{event_id}_i, t_i, c_i, o_i)$
	return t_i

Note that \mathcal{A} always solves OMDL if \mathcal{F} breaks unforgeability except in the case the condition in line 4 causes \mathcal{A} to abort. This can only occur if the forger has created an `event_id` with two different possible outcomes that hash to the same c which implies \mathcal{F} has broken the collision resistance of H . We can therefore bound the probability of \mathcal{F} succeeding by the difficulty of breaking collision resistance and OMDL and write $\varepsilon \leq \varepsilon' + \varepsilon_{\text{cr}}$. Since our reductions takes whatever time \mathcal{F} takes plus the time taken to respond to \mathcal{F} 's queries we write $\tau_{\text{cr}} = \tau + O(n)$ and $\tau' = \tau + O(n)$. \square

Apart from the security result itself the important thing to note is that it was not necessary to model the hash function H as a random oracle to prove security as it is with the ordinary Schnorr scheme [3]. The hash function is required to be collision resistant but upon closer inspection it appears not to need to be a hash function at all; a collision free mapping from an outcome to an element of \mathbb{Z}_q would suffice. This means that simply ordering the the outcomes from o_1, \dots, o_m and setting $c = j$ for o_j would yield a simpler and secure scheme that does not depend on the collision resistance of any hash function. The only apparent downside is that the attestation when combined with the announcement would no longer be a valid Schnorr signature but it is unclear how useful that feature is.

References

1. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. Cryptology ePrint Archive, Report 2001/002 (2001), <https://eprint.iacr.org/2001/002>

2. Dryja, T.: Discreet log contracts. <https://adiabat.github.io/dlc.pdf> (2017)
3. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. *Journal of Cryptology* **13**(3), 361–396 (2000)