Security of Discreet Log Contract Attestation Schemes

Lloyd Fournier

lloyd.fourn@gmail.com

1 Introduction

In *Discreet Log Contracts*[3], 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 formally covers the notion of attestation unforgeability.

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

- $\mathsf{KeyGen}(\cdot) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$*: A key generation algorithm*
- $\mathsf{Announce}(\mathsf{sk}, \mathtt{id}) \xrightarrow{\$} (r, a)$*: An event announcement algorithm which the oracle uses to generate an announcement secret* r *and an announcement* a*.*
- $\mathsf{Attest}(\mathsf{sk}, \mathtt{id}, r, a, o) \to t$*: An event outcome attestation algorithm which the oracle uses to publicly attest to the fact that an event identified previously announced with* a *has has the outcome* o*.*
- $\mathsf{Anticipate}(\mathsf{pk}, \mathtt{id}, a, o) \to T$*: A attestation anticipation scheme which returns an image* T *of the eventual attestation for a particular outcome* o*.*
- $\mathsf{Verify}(\mathsf{pk}, \mathtt{id}, a, t, o) \to \{0, 1\}$*: 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*.*

The workflow of an oracle attestation scheme starts with the oracle invoking Announce and publishing the resulting announcement. Users input the announcement into Anticipate to produce a (protocol specific) image of the eventual attestation, for example, a group element $T = tG$ where t would be the final attestation for a particular outcome. They use this to condition the release of different quantities of coins to each other depending on which value of t is finally revealed. When the oracle decides the outcome o of the event, they compute the corresponding attestation t and publish (o, t). The users then use t to release the coins they are due upon o being the outcome. Since we do not actually model the on-chain betting protocol we use Verify to capture attestation validity since it is convenient for our security definitions. In practice users would not necessarily ever execute Verify.

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*. In the real world we assume that announcements are signed under a different key (also known to belong to the oracle) so that each user can verify an announcement did indeed come from an oracle before using it. Forcing announcements to be signed under the attestation key would needlessly complicate the security analysis since it would also give us the extra task of proving that the announcement signature scheme remains unforgeable when composed with an attestation scheme. 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 context of a security game definition.

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

Forge

$i \leftarrow 1; Q_{\mathsf{Ann}} := \emptyset; Q_{\mathsf{Att}} := \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow_{\$} \mathsf{KeyGen}$
$(i^*, a^*, t^*, o^*) \leftarrow_{\$} \mathcal{F}^{\mathsf{Ann},\mathsf{Att}}(\mathsf{pk})$
return
$\exists (i^*, \mathtt{id}^*, \cdot, a^*) \in Q_{\mathsf{Ann}} \wedge$
$\exists (\mathtt{id}^*, o) \in Q_{\mathsf{Att}} \wedge$
$o^* \neq o \wedge$
$\mathsf{Verify}(\mathsf{pk}, \mathtt{id}^*, a^*, t^*, o^*)$

Oracle $\mathsf{Ann}(\mathtt{id}_i)$

$(r_i, a_i) \leftarrow_{\$} \mathsf{Announce}(\mathsf{sk}, \mathtt{id}_i)$
$Q_{\mathsf{Ann}} := Q_{\mathsf{Ann}} \cup \{(i, \mathtt{id}_i, r_i, a_i)\}$
$i \leftarrow i + 1$
return a_i

Oracle $\mathsf{Att}(\mathtt{id}_i, o_i)$

check $\exists (i, \mathtt{id}_i, r_i, a_i) \in Q_{\mathsf{Ann}} \wedge (\mathtt{id}_i, \cdot) \notin Q_{\mathsf{Att}}$
$t_i \leftarrow \mathsf{Attest}(\mathsf{sk}, \mathtt{id}_i, r_i, a_i, o_i)$
$Q_{\mathsf{Att}} := Q_{\mathsf{Att}} \cup \{(\mathtt{id}_i, o_i)\}$
return t_i

Without loss of generality we assume that $\mathcal{F}$ always queries Att *with an* $\mathtt{id}$ *if it has queried* Ann *with it before returning its forgery. However it may do so in any order. We allow the outcome o queried to* Att *to be any string.*

Note that in reality o cannot be any string but must be part of a set determined by the $\mathtt{id}$. Primarily, we give the adversary the extra power to choose an arbitrary string because it is simpler not to have to reason about how the constrained outcome space affects the analysis. Secondarily, it extends the security model's guarantees to an attestation hardware device that cannot verify what is and is not a semantically valid outcome for a particular $\mathtt{id}$.

3 Dryja's Scheme

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.

KeyGen	Announce($\mathsf{sk}, \mathtt{id}$)	Attest($\mathsf{sk}, \mathtt{id}, r, a, o$)
$x \leftarrow_{\$} \mathbb{Z}_q$	$r \leftarrow_{\$} \mathbb{Z}_q$	$(x, X) := \mathsf{sk}; R := a$
$X \leftarrow xG$	$R \leftarrow rG$	$c \leftarrow H(R, X, \mathtt{id}, o)$
$\mathsf{sk} := (x, X); \mathsf{pk} := X;$	**return** (r, R)	$t \leftarrow r + cx$
return $(\mathsf{sk}, \mathsf{pk})$		**return** t

Anticipate($\mathsf{pk}, \mathtt{id}, a, o$)	Verify($\mathsf{pk}, \mathtt{id}, a, t, o$)
$X := \mathsf{pk}; R := a$	$X := \mathsf{pk}; R := a$
$c \leftarrow H(R, X, \mathtt{id}, o)$	$c \leftarrow H(R, X, \mathtt{id}, o)$
return $R + cX$	**return** $R = tG - cX$

Fig. 1. Dryja's Schnorr-like oracle attestation scheme defined for a group with generator G of prime order q.

This pre-commitment of R is a major deviation from typical Schnorr signatures. In the standard proof of *existential unforgeability under chosen message attack* [7] the message is presented to the honest signer before it chooses the nonce R. Unfortunately, this means we cannot use the existing unforgeability of Schnorr signatures to imply to the attestation unforgeability of Dryja's oracle attestation scheme. Instead we show a new reduction from the *one-more discrete logarithm problem* (OMDL) to attestation unforgeability in the plain model.

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 DLog_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)-solve the n-OMDL problem in $\mathbb{G}$ if it runs in at most time τ, makes at most n queries to DLog_G and $\Pr[n{-}\mathsf{OMDL} = 1] \geq \varepsilon$ for the $n{-}\mathsf{OMDL}$ experiment below.

$n{-}\mathsf{OMDL}$

$x_0, \ldots, x_n \leftarrow_\$ \mathbb{Z}_q^{n+1}$
for $i = 0, \ldots, n$ **do**
  $X_i \leftarrow x_i G$
$x_0^*, \ldots, x_n^* \leftarrow_\$ \mathcal{A}^{\mathsf{DLog}_G}(X_0, \ldots, X_n)$
return $(x_0^*, \ldots, x_n^*) = (x_0, \ldots, x_n)$

Theorem 1 ($n{-}\mathsf{OMDL}$ 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')-solves the n-OMDL problem and an adversary that $(\tau_{\mathsf{cr}}, \varepsilon_{\mathsf{cr}})$-breaks the collision resistance of H such that*

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

Proof. We use a forger $\mathcal{F}$ against the unforgeability of Dryja's scheme to construct an adversary $\mathcal{A}$ against $n{-}\mathsf{OMDL}$. We take the first challenge group element X_0 and use it as our oracle's public key. Our strategy is to then to use the remaining elements $X_1, \ldots, X_n$ as the announcements from the Ann oracle and then use the DLog_G oracle to simulate the Att oracle. 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.

$\mathcal{A}^{\mathsf{DLog}_G}(X_0, ..., X_n)$

1 : $i \leftarrow 1$
2 : $(i^*, a^*, t^*, o^*) \leftarrow_\$ \mathcal{F}^{\mathsf{Ann},\mathsf{Att}}(X_0)$
3 : $(\mathtt{id}^*, t, c, o) := Q_{i^*}$
4 : $c^* \leftarrow H(X_{i^*}, X_0, \mathtt{id}^*, o^*)$
5 : **if** $c = c^*$ **then abort**
6 : $x_0 \leftarrow (t - t^*)/(c - c^*)$
7 : **for** $j = 1, \ldots, n$ **do**
8 : $(\mathtt{id}_j, t_j, c_j, o_j) := Q_j$
9 : $x_j \leftarrow t_j - c_j x_0$
10 : **return** $(x_0, \ldots, x_n)$

Simulate $\mathsf{Ann}(\mathtt{id}_i)$

$a_i := X_i$
$i \leftarrow i + 1$
return a_i

Simulate $\mathsf{Att}(\mathtt{id}_i, o_i)$

$c_i \leftarrow H(X_i, X_0, \mathtt{id}_i, o_i)$
$S_i \leftarrow X_i + cX_0$
$t_i \leftarrow \mathsf{DLog}_G(S_i)$
$Q_i := (\mathtt{id}_i, t_i, c_i, o_i)$
return t_i

Note that $\mathcal{A}$ always solves $n{-}\mathsf{OMDL}$ if $\mathcal{F}$ breaks unforgeability except in the case the condition in line 5 causes $\mathcal{A}$ to abort. This can only occur if the

forger finds two $\mathtt{id}$ and o pairs 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 in addition to $n{-}\mathsf{OMDL}$ and write $\varepsilon \leq \varepsilon' + \varepsilon_{\mathsf{cr}}$. Since our reductions takes whatever time $\mathcal{F}$ takes plus the time taken to respond to $\mathcal{F}$'s queries we write $\tau_{\mathsf{cr}} = \tau + O(n)$ and $\tau' = \tau + O(n)$. □

It is important to note that the extra collision resistance requirement is not strictly necessary because it is an artefact of giving the adversary the power to choose any arbitrary string as the outcome after seeing the announcement. If we remove this power from the adversary then the scheme is likely secure under a weaker hash function requirement.

4 A Simpler Scheme

Observe that in the proof in the previous section 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 [7]. Upon closer inspection it appears producing c need not be done with a hash function at all; any injective (collision-fee) mapping from outcomes to to $\mathbb{Z}_q$ would suffice. The most convenient modification would replacing each invocation of the hash $H(R, X, \mathtt{id}, o)$ with $\mathsf{ord}(\mathtt{id}, o)$ in Figure 1 where ord simply returns the index of the outcome o in $\mathtt{id}$ e.g. 0 for the first outcome and 1 for the second etc. Crucially, the proof for Theorem 1 would hold as all that is required is to guarantee unforgeability is that $c \neq c^*$ when $o \neq o^*$. This is clearly unconditionally guaranteed by setting $c \leftarrow \mathsf{ord}(\mathtt{id}, o)$.

$\mathsf{Attest}(\mathsf{sk}, \mathtt{id}, r, a, o)$	$\mathsf{Verify}(\mathsf{pk}, \mathtt{id}, a, t, o)$
$(x, X) := \mathsf{sk}; R := a$	$X := \mathsf{pk}; R := a$
$c \leftarrow \mathsf{ord}(\mathtt{id}, o)$	$c \leftarrow \mathsf{ord}(\mathtt{id}, o)$
$t \leftarrow r + cx$	**return** $R = tG - cX$
return t	

Fig. 2. A simplification of the attestation scheme in Figure 1 where the hash function is replaced by ord which returns the index of the outcome in the event.

The only apparent downside of this idea is that the attestation when combined with the announcement would no longer be a valid Schnorr signature. However, as we have mentioned the pre-commiting of the nonce breaks the unforgeability proofs for Schnorr so a user must always very the attestation with respect to an announcement and never verify it as a plain Schnorr signature.

5 Oracle Aggregation Security

An obvious question is whether a user can securely combine n oracles into an aggregate oracle such that they can only obtain an attestation from the aggregate oracle if each component oracle attests to the particular outcome on a particular event. The user should be able to do this without the oracles interacting with each other or even being aware of each other. The most straightforward reason to do this is to make sure several oracles agree on the outcome of an event so that one corrupt oracle is not enough to affirm the outcome. Furthermore it could be useful to combine unrelated events into a single cryptographic condition. For example, Bob may wish to combine the conditions of "The Democrats will win the election and it will be raining in Shanghai on March 1st". We wish to preserve security if he uses the same oracle for both events or different ones.

To incorporate this functionality we extend the definition in Figure 2 with the following algorithms:

- $\mathsf{AggAnticipate}((\mathsf{pk}_1, \mathtt{id}_1, a_1, o_1), \ldots, (\mathsf{pk}_n, \mathtt{id}_n, a_n, o_n)) \to T$: A deterministic aggregate anticipation algorithm that returns the anticipated attestation for a set of announcements.
- $\mathsf{AggAttest}(t_1, \ldots, t_n) \to t$: A deterministic attestation combination algorithm which combines a set of attestations into an aggregate attestation t.

In [3] it is suggested to extend the scheme in Figure 1 by simply adding the anticipation points and attestation scalars from single oracle schemes as shown in Figure 5.

$\underline{\mathsf{AggAnticipate}((\mathsf{pk}_1, \mathtt{id}_1, a_1, o_1), \ldots, (\mathsf{pk}_n, \mathtt{id}_n, a_n, o_n))}$

return $\Sigma_{i=1}^{n} \mathsf{Anticipate}(\mathsf{pk}_i, \mathtt{id}_i, a_i, o_i)$

$\underline{\mathsf{AggAttest}(t_1, \ldots, t_n)}$

return $\Sigma_{i=1}^{n} t_i$

Fig. 3. The oracle aggregation scheme from [3].

Unfortunately the scheme above is insecure as it was originally presented. A malicious oracle after seeing another oracle's announcement may contrive their own announcement to *cancel out* the attestation of the other so that oracle alone has the power to settle the bet. Observe that if the honest oracle's anticipated attestation on an outcome o_1 is R_1 and their anticipation point is $T_1 = R_1 + H(R_1||X_1||\mathtt{id}||o_1)X_1$ then a malicious oracle with key X_1 can set their announcement $R_2 = R_2^{'} - T_1$ where $R_2^{'} = r_2^{'}G$ for some randomly selected $r_2^{'}$. This cancels out the honest oracle's contribution to attestation completely as follows:

$$\begin{aligned}\mathsf{AggAnticipate}((X_1, \mathtt{id}_1, R_1, o_1), (X_2, \mathtt{id}_2, R_2, o_2)) &= T_1 + T_2 \\ &= R_2^{'} + H(R_2||X_2||\mathtt{id}_2||o_1)X_2\end{aligned}$$

Clearly the the malicious oracle can now reveal the combined attestation as $r_2' + H(R_2||X_2||id_2||o_2)x_2$. To address this problem we can force oracles to prove knowledge of the the secret nonce in the announcement. Note carefully that the malicious oracle does not know the secret key for their announced nonce R_2, in the above example and so would be unable to produce the proof. We conjecture that in the *knowledge of secret key* model (KOSK) [2] this would allow us to prove the oracle aggregation scheme to be attestation unforgeable. We leave it to a future revision to prove this conjecture.

6 Multi-Party Attestation

An oracle may wish to split up the responsibility of attesting under its public key to multiple parties so that they all must be corrupted to get the oracle to attest to an invalid outcome. For example, one key may be given to a system that uses the web to scrape the web to produce the outcome while another may require a human to manually sign the outcome. To the outside world this oracle should appear as a single oracle.

The most straightforward way to do this with Dryja's scheme would be to take each party's key $X_1, \ldots, X_n$ and aggregate them using a Schnorr multi-signature scheme to get the combined oracle's key. Unfortunately, just as in the single oracle case, doing this breaks the unforgeability proof of most existing Schnorr multi-signature schemes. The reason for this is that each party must also generate their component of the announcement and aggregate them into a single announcement. In the case of Dryja's scheme this means declaring the Schnorr nonce R up front by computing it somehow from the component nonces. This breaks the unforgeability proof for for example MuSig[4] because it requires that the adversary does not know R before the choosing the message to receive a signature on. Although MuSig2[5] actually does allow pre-generating public nonce data in the case of multi-signatures it does not allow computing the final nonce R until the message is known and so cannot be applied here either. We note that the robust MuSig-DN[6] would circumvent this problem and would work at the cost of its complexity.

Just as in the single oracle case it is likely the theoretical problems with the MuSig proof could be remedied by forcing the adversary to fix the list of outcomes at announcement time. We will not dwell on this point because the fact that signatures could be forged does not actually mean it is an insecure multi-party attestation scheme. In a future revision we hope to formally prove the security of a multi-party attestation scheme without reference to Schnorr multi-signature security.

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. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: International Workshop on Public Key Cryptography. pp. 31–46. Springer (2003)
3. Dryja, T.: Discreet log contracts. https://adiabat.github.io/dlc.pdf (2017)
4. 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
5. Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round schnorr multi-signatures. Cryptology ePrint Archive, Report 2020/1261 (2020), https://eprint.iacr.org/2020/1261
6. Nick, J., Ruffing, T., Seurin, Y., Wuille, P.: Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. Cryptology ePrint Archive, Report 2020/1057 (2020), https://eprint.iacr.org/2020/1057
7. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3), 361–396 (2000)