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:*

- $\mathsf{KeyGen}(\cdot) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$*: A key generation algorithm*
- $\mathsf{Announce}(\mathsf{sk}, \texttt{event\_id}) \xrightarrow{\$} (r, a)$*: An event announcement algorithm which the oracle uses to generate an announcement secret* r *and an announcement* a*.*
- $\mathsf{Attest}(\mathsf{sk}, \texttt{event\_id}, r, a, o) \rightarrow t$*: An event outcome attestation algorithm which the oracle uses to publicly attest to the fact that an event identified previously announced with* a *has has the outcome* o*.*
- $\mathsf{Verify}(\mathsf{pk}, \texttt{event\_id}, a, t, o) \rightarrow \{0, 1\}$*: 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 $\mathcal{F}$ (τ, ϵ, 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* $\Pr[\mathsf{Forge} = 1] \geq \epsilon$ *in the experiment defined below.*

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

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

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

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. We then go on to show that the scheme is secure if the *one-more discrete logarithm problem* is hard by reduction.

```
KeyGen
    $x \leftarrow_{\$} \mathbb{Z}_q$
    $X \leftarrow xG$
    $\mathsf{sk} := (x, X); \mathsf{pk} := X;$
    $\mathbf{return}\ (\mathsf{sk}, \mathsf{pk})$

Announce(sk, event_id)
    $r \leftarrow_{\$} \mathbb{Z}_q$
    $R \leftarrow rG$
    $\mathbf{return}\ (r, R)$

Attest(sk, event_id, $r, a, o$)
    $(x, X) := \mathsf{sk}; R := a$
    $c \leftarrow H(R, X, \texttt{event\_id}, o)$
    $t \leftarrow r + cx$
    $\mathbf{return}\ t$

Verify(pk, event_id, $a, t, o$)
    $X := \mathsf{pk}; R := a$
    $c \leftarrow H(R, X, \texttt{event\_id}, o)$
    $\mathbf{return}\ R = tG - cX$
```

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

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 $\Pr[n\text{−}\mathsf{OMDL} = 1] \geq \varepsilon$ for the $n\text{−}\mathsf{OMDL}$ 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 :  $(i^*, t^*, o^*) \leftarrow_{\$} \mathcal{F}^{\mathsf{Ann},\mathsf{Att}}(X_0)$
3 :  $(\texttt{event\_id}^*, t, c, o) := Q_{i^*}$
4 :  $c^* \leftarrow H(X_{i^*}, X_0, \texttt{event\_id}^*, o^*)$
5 :  if $c = c^*$ then abort
6 :  $x_0 \leftarrow (t - t^*)/(c - c^*)$
7 :  for $j = 1, \ldots, n$ do
8 :      $(\texttt{event\_id}_j, t_j, c_j, o_j) := Q_j$
9 :      $x_j \leftarrow t_j - c_j x_0$
10 : return $(x_0, \ldots, x_n)$
```

```
Simulate $\mathsf{Ann}(\texttt{event\_id}_i)$
$a_i := X_i$
$i \leftarrow i + 1$
return $a_i$
```

```
Simulate $\mathsf{Att}(\texttt{event\_id}_i, o_i)$
$c_i \leftarrow H(X_i, X_0, \texttt{event\_id}_i, o_i)$
$S_i \leftarrow X_i + cX_0$
$t_i \leftarrow \mathsf{DLog}_G(S_i)$
$Q_i := (\texttt{event\_id}_i, t_i, c_i, o_i)$
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 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 in addition $n-\mathsf{OMDL}$ 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 $\tau' = \tau + O(n)$. □

We note that technically collision resistance is not strictly necessary to prove security. Reaching the collision condition is intractable since $\mathcal{F}$ has to choose the `event_id` and therefore the possible values of o before querying Ann. The oracle then gets to randomly choose the prefix of pre-image (R in Figure 1) which naturally thwarts attempts to contrive a set of outcomes that results in a collision. We leave clearing up this technicality to future revision.

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 [3]. Upon closer inspection it appears producing c need not be done with a hash function at all; any collision free mapping from an outcome to an element of $\mathbb{Z}_q$ would suffice. The most convenient modification would replacing each invocation of the hash $H(R, X, \texttt{event\_id}, o)$ with $\mathsf{ord}(\texttt{event\_id}, o)$ in Figure 1 where ord simply returns the index of the outcome o in $\texttt{event\_id}$ 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 $c \leftarrow \mathsf{ord}(\texttt{event\_id}, o)$.

```
$\mathsf{Attest}(\mathsf{sk}, \texttt{event\_id}, r, a, o)$
$(x, X) := \mathsf{sk}; R := a$
$c \leftarrow \mathsf{ord}(\texttt{event\_id}, o)$
$t \leftarrow r + cx$
**return** $t$

$\mathsf{Verify}(\mathsf{pk}, \texttt{event\_id}, a, t, o)$
$X := \mathsf{pk}; R := a$
$c \leftarrow \mathsf{ord}(\texttt{event\_id}, o)$
**return** $R = tG - cX$
```

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. It is unclear how useful that feature is in practice.

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)