This post provides a high-level overview of DPACCs, a commitment scheme which can be built on top of existing smart contract wallet (SCW) architecture. DPACCs allow any SCW holder to collateralize an on-chain commitment in a private and anonymous manner. DPACCs can prove arbitrarily much or little about the wallet generating the commitment, and/or the transaction which is being committed. This can be used to convince a prospective block builder or relayer that the wallet generating the DPACC has enough funds to pay required fees, that the wallet is committed to performing certain actions, and importantly, that the wallet loses some amount of collateral if this commitment is broken. DPACCs delegate typically expensive zero knowledge operations off-chain, only requiring an additional one or two mapping checks when compared to transactions being sent from basic externally owned accounts.

ZK Primer

For this article, we need a function that allows us to prove set-membership in zero knowledge. In the following h()

is a hash function, and ||

is string concatenation. We define that function as follows. For a set of commitments \textit{Commits}

, NIZKPoK(\mathcal{S},\mathcal{R},\textit{Commits}

) returns a string \mathcal{S}

and non-interactive proof of knowledge that the person running NIZKPoK

() knows an \mathcal{R}

such that the commitment h(\mathcal{S} || \mathcal{R})

is in \textit{Commits}

. Importantly,

- \mathcal{R}

is not revealed, so the verifier of the proof does not learn which commitment in \textit{Commits}

the proof corresponds to.

- This revelation identifies to a verifier when a proof has previously been provided for a particular, albeit unknown, commitment as the prover must reproduce \mathcal{S}

.

There are many tools to achieve this, including those used inTornado Cash, Semaphore, ZCash, and others.

DPACCs

It suffices to consider SCWs as extensions of externally-owned accounts on which we can apply additional constraints. For W

the set of all wallets, and some wallet w \in W

, w

can be considered as a set of tokens. To reason about DPACCs, we need two mappings.

- For every SCW w

, a secret commitment mapping

C^w_S: {0,1}^{\Theta(\kappa)} \rightarrow B \subseteq w

. C^w_S(x)

is such that if x \neq y

, C^w_S(x) \cap C^w_S(y) = \emptyset

.

- A global transaction commitment mapping

C_T: {0,1}^{\Theta(\kappa)} \rightarrow {0,1}^{\Theta(\kappa)}

.

Players in our system using SCWs maintain a mapping of secret commitments to tokens within their wallet. To use tokens in a wallet, users must provide a signed transaction, as in a basic externally-owned account, and the secret corresponding to a commitment/commitments within the wallet. Revealing the secret $\mathcal{S}$

corresponding to a non-zero commitment $\textit{com}=h(\mathcal{S}||\mathcal{R})$

forces w

to use only tokens in $C^w_S(\textit{com})$

. At the end of a transaction, the user must generate new mappings for all unmapped tokens in the SCWs, with the default being the 0-secret commitment. If $C^w_S(\textit{com}=h(\mathcal{S}||\mathcal{R}))=X$

for some set of tokens X

, X

is fixed (tokens cannot be added or removed from X

) until $\mathcal{S}$

is revealed.

A further restriction on SCWs is the verification that a revealed secret corresponds to a valid entry in the global transaction commitment mapping. At initialization, $C_T(x)=0, \ \forall x \in \{0,1\}^{\Theta(\kappa)}$

. When a SCW w

signs a transaction tx

and attempts to use tokens $X\subseteq C^w_S(\mathcal{S})$

, it must be that either $C_T(\mathcal{S}) = 0$

, or $C_T(\mathcal{S}) = h(tx)$

.

Otherwise, the transaction is aborted. To add a mapping $C_T: x \rightarrow y$

, we require that $C_T(x)=0$

and y

is provably signed by the player who generated x

using some pre-defined PKI.

Example of how DPACCs can thwart MEV

Consider a set of wallets $\{w_1,...,w_n\}=W$

with non-zero secret commitment mappings to their token balances. That is, any wallets in W

must reveal the committed secret in order to use their tokens balances. SCWs integrated to use DPACCs can transact as normal, revealing the commitment secret corresponding to tokens, and then using these tokens for normal transactions, as is done with externally owned accounts. This requires first verifying the mapping in the SCW, then verifying that the secret does not correspond to a mapping in the global transaction commitment mapping. Two mapping checks are cheap (much less than 20k gas, the cost for inserting to mapping although not sure exactly).

DPACCs become interesting when one of the SCWs wants to insert a DPACC into the global transaction commitment mapping.

Auctions demonstrate the power of DPACCs. Consider an auction occurring where buyers and sellers are allowed to join until a fixed cut-off. Revealing order information before all players have committed to their orders is an elementary source of MEV. Instead of sending plain-text transactions, users can send DPACCs.

Let’s say a user u_i

wanting to sell 1 ETH for USDC creates a sell order for 1 ETH. The user generates a commitment for the transaction $tx=\textbf{pay fee to relayer}|| \textbf{sell 1 ETH for USDC}$

.

The users generates two proofs to be sent with h(tx)

to a relayer pool:

1. NIZKPoK(\textbf{pay fee to relayer}, \textbf{sell 1 ETH for USDC},{h(tx)}

). Recalling the primer on ZK subsection, this proves that the prefix of the transaction committed by h(tx)

, which the relayer can see, pays a relayer a fee if it is included in the blockchain, but doesn't reveal anything about the rest of tx

.

1. NIZKPoK(\mathcal{S}, \mathcal{R},\textit{Commits}

), for \textit{Commits}

being the set of secret commitments of wallets with balances of at least 0.1

ETH or 100

USDC, after fees. Any relayer with access to the chain state can verify if this proof is correct. Specifically, the verifier knows that if \mathcal{S} \rightarrow h(tx)

is added to the global transaction commitment mapping, u_i

must reveal tx

, or lose at least 0.1

ETH or 100

USDC if the transaction is not revealed. In reality, the user in question loses at least 1

ETH, but this is not important to the relayer.

If h(tx)

is added to the transaction commitment mapping, the next transaction from u_i

must be tx

. This is because u_i

must reveal \mathcal{S}

to use the tokens in her wallet. As \mathcal{S}

maps to h(tx)

, the only valid transaction that can be executed is tx

. Therefore, if the auction contract punishes late revelation, as u_i

committed to an order in the auction, u_i

is effectively forced to reveal this order. If all orders must be committed before the revelation phase, the entire auction consists of committed orders which reveal nothing consequential about the orders, or the players who committed to them.

As such, DPACCs allow integrated SCW users to join any ongoing auction, whether it be a DEX auction (seebatch auctions and their price guarantees under competition), NFT auction, or liquidation auction, all without being subjected to MEV

.

Other use cases include requests-for-quotes, where users commit to buying or selling a token if a liquidity provider (LP) includes the commitment in the blockchain along with a market with bid equal to offer price. If the user commits to paying the liquidity provider a fee which is increasing in block number, this effectively creates a dutch auction among LPs, which should give users near optimal pricing. This idea resembles fee-escalators, with the crucial benefit of hiding meaningful order information until the LP has committed to a price.

This post is based on a technical paper which will be linked when uploaded to Arxiv. Thanks in advance for any thoughts,

feedback, help, and/or ideas for collaboration around DPACCs.