ZEX: Confidential Peer-to-Peer DEX

v0.1

Artem Chystiakov, Mariia Zhvanko
Distributed Lab

August 2025

1 Introduction

Ethereum's transparency, while being one of its most valuable advantages, has become a barrier to real-world financial adoption. Public blockchains expose balances, approvals, counterparties, and even token acquisitions by default.

The proposal introduces ZEX, a mechanism for permissionless and confidential peer-to-peer Decentralized Exchange (DEX) using cWETH [1] as a model for confidential tokens. It extends the token functionality with confidential allowances, enabling hidden peer-to-peer interactions. Similar to the cWETH protocol, ZEX logic utilizes Elliptic Curve (EC) Twisted ElGamal-based commitment scheme to ensure confidentiality, EC Diffie-Hellman (DH) to introduce value accessibility, and zk-SNARKs to verify the correctness of balance updates during token approvals and transfers without revealing the actual amounts exchanged.

The ZEX protocol serves as a marketplace for swap offers where exchanges happen in a three-step interactive fashion and require multiple ZK proofs to be verified within the same transaction. The only piece of information disclosed by ZEX is the price and initial offer allowance, indicating the maximum amount of tokens available for the particular exchange.

Modifications of the protocol may also conceal the swap price to increase privacy assumptions.

2 Confidential Token Allowance Model

In order to perform a confidential peer-to-peer swap, the approve and transferFrom flows must be defined over the confidential token.

These flows will be managed by the six new functions in total:

1. *confidentialApprove* to approve tokens to an EOA with existing encryption keys without disclosing the approved amount.

2. *confidentialTransferFrom* to confidentially transfer approved tokens without disclosing the transferred amount.

3. *publicConfidentialApprove* to one-time approve tokens to a smart contract, disclosing the approved amount.

4. *publicConfidentialTransferFrom* to transfer publicly approved tokens without disclosing the transferred amount.

5. *cancelConfidentialApprove* to cancel the confidential approval without disclosing the canceled amount.

6. *cancelPublicConfidentialApprove* to cancel the public approval, disclosing the canceled amount.

For the in-depth specification of the cryptography used by these functions, please refer to the original cWETH [1] paper.

2.1 Confidential approve & transferFrom

Contrary to the regular ERC-20 approval logic, confidential approve results in a decrease in the approver's balance. This is because during the confidential transferFrom process, the spender cannot verify if the approver has sufficient balance, as it is encrypted.

Additionally, the confidential approve operation requires the operator address to be provided, which is supplementary to the usual ERC-20 spender's address and allowance amount. If the operator address is specified, then the spender cannot call confidentialTransferFrom to receive tokens; only the operator is eligible to do so. This constraint is crucial for the confidential transferFrom security and customization, as it enables the approver to specify a smart contract as an operator to enforce additional transfer validation.

The confidential approve to the EOA flow is depicted in the following diagram:

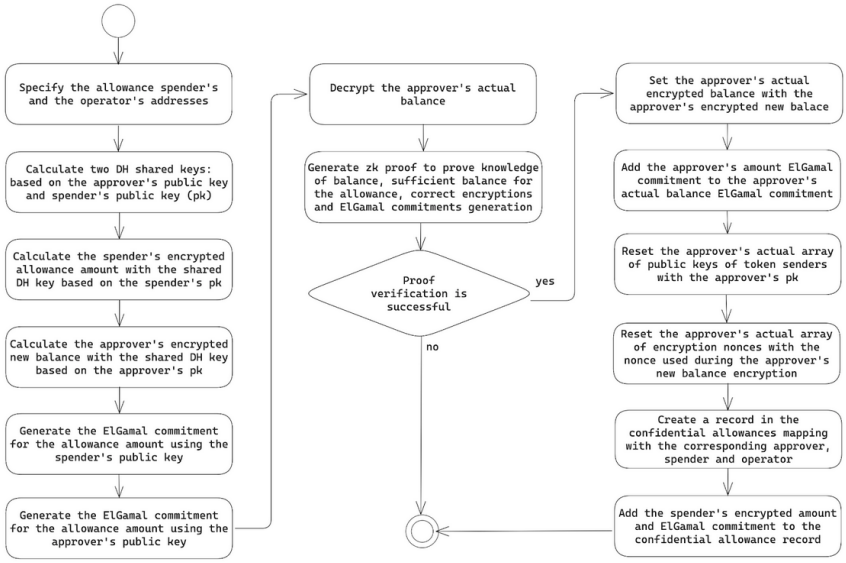

Figure 1: Confidential approve flow

Once the allowance has been granted, the spender is able to receive approved tokens. The diagram below illustrates the confidential transferFrom process:

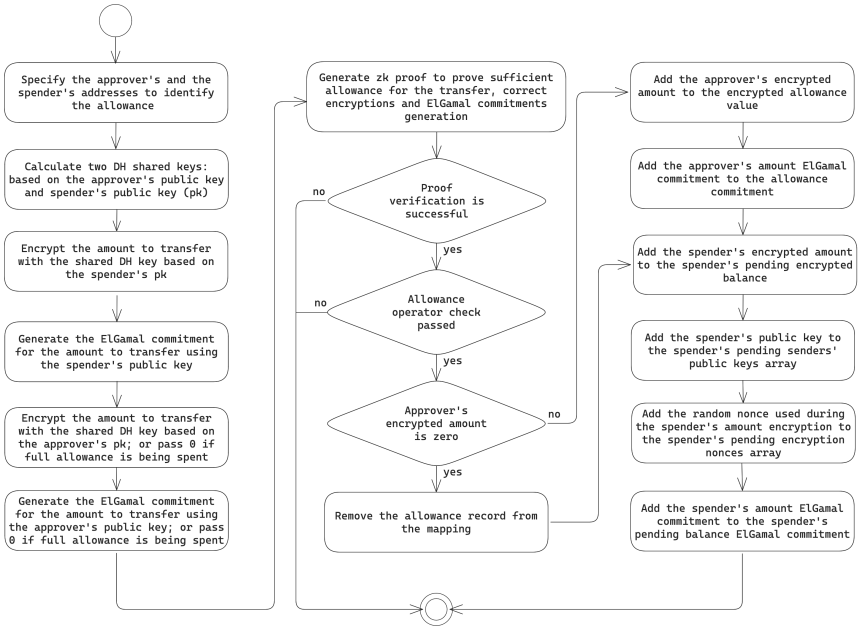

Figure 2: Confidential transferFrom flow

It is crucial to understand that the confidential approve and transferFrom operate on five different encrypted amounts:

1. Approver's self-ECDH key to represent their new balance after the allowance is granted.
2. Approver's ElGamal allowance commitment for them to be able to access the allowance amount in case of approval cancellation. This commitment is also subtracted from the approver's balance commitment to represent the new balance after allowance is granted.
3. Allowance amount encrypted with the approver/spender ECDH shared key for the allowance amount availability.
4. Spender's ElGamal allowance commitment that is added to their pending balance upon allowance spending.
5. The received token amount encrypted with a self-ECDH key for the new spender balance availability after the allowance is spent.

2.2 Public confidential approve & transferFrom

In some scenarios, it is needed to approve tokens not to an EOA with encryption keys, but directly to a smart contract. Since smart contracts don't hold a confidential private key, they cannot decrypt the confidential allowance values. Moreover, in this case, the approver does not know the receiver beforehand and cannot encrypt the value for the specific public key. As a result, any confidential allowance intended for a smart contract must disclose the approved amount publicly.

The diagram below depicts the public confidential approve flow:

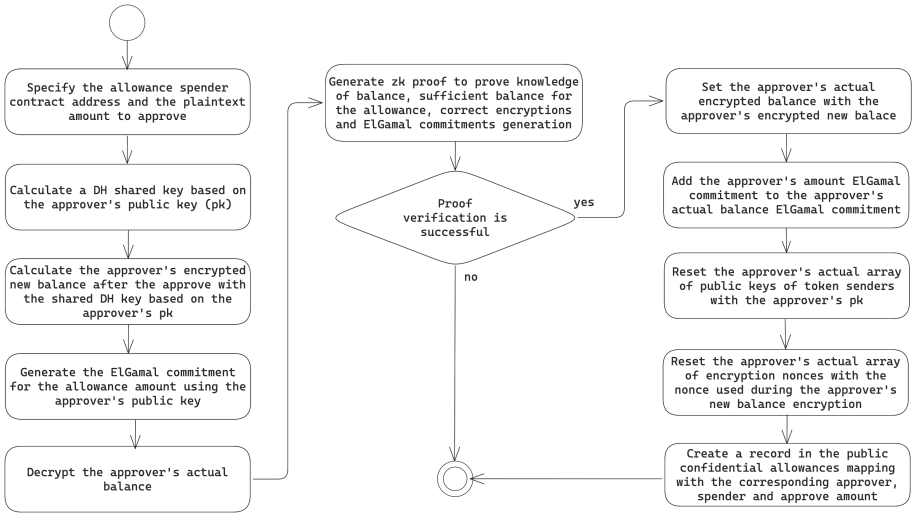

Figure 3: Public confidential approve flow

Because the approved amount is specified in plaintext, a public confidential allowance must be spent entirely and in an atomic fashion, making it a one-time approval. In other words, if the public allowance is not being spent fully, the privacy assumptions require the remaining value to be re-encrypted with the approver's public key and added back to the approver's pending balance in the same transaction. This is caused by the fact that the public update of the allowance value will result in compromising the confidentiality of the amounts being transferred.

The flow of spending the public confidential allowance is depicted in the following diagram:

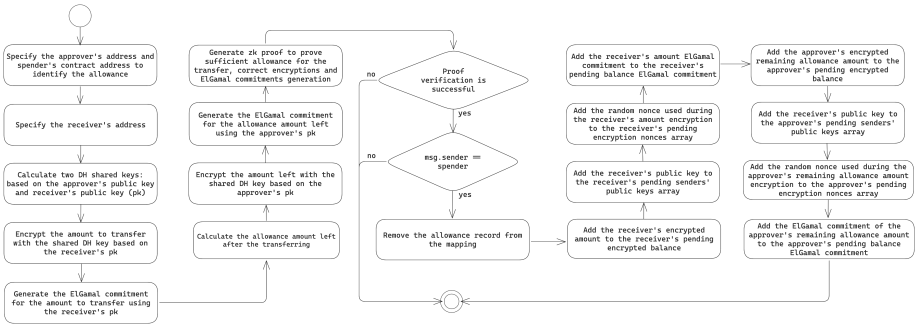

Figure 4: Public confidential transferFrom flow

The significant difference with the regular confidential transferFrom is that the allowance leftover gets encrypted and added back to the approver's pending balance, instead of residing in the dedicated allowance storage.

2.3 Allowance cancellation

The protocol also includes a mechanism for an approver to cancel a confidential allowance, whether it was granted as an encrypted allowance to an EOA or a public allowance to a contract.

The confidential allowance cancellation flow is shown in the diagram below:

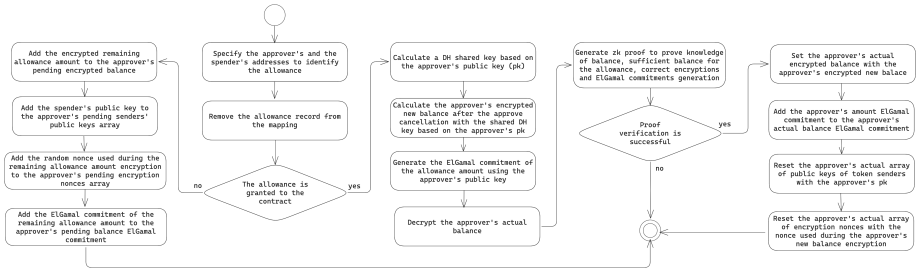

Figure 5: Allowance cancellation flow

The confidential allowance cancellation flow is rather straightforward since all the encrypted amounts are being maintained in the confidential approve and transferFrom operations. All that is required is to merely add them back to the approver's pending balance.

However, the public cancellation additionally requires a ZK proof for the verification of the encryption of the allowance leftover.

3 ZEX Protocol Overview

3.1 Prerequisites

For the DEX to operate correctly, several conditions must be met to keep the swap process smooth and consistent:

- All participating parties must have their confidential public keys published. These keys are used for both EC ElGamal commitments and ECDH encryption, following the scheme used in the underlying confidential tokens.
- The offer initiator must hold enough confidential tokens in the sell asset to cover the maximum amount they intend to make available in the offer.

- The offer acceptor must hold enough confidential tokens in the buy asset to pay the agreed amount according to the specified rate.
- The offer acceptor must have access to the offer details before preparing their part of the swap. Specifically, the offer initiator must publicly publish the exchange rate and the maximum available amount of the asset to sell.

3.2 Peer-to-peer swap flow

The following section describes the execution flow of the confidential peer-to-peer swap powered by the confidential allowance model previously defined. The exchange process is interactive and consists of three stages: *offer placement*, *offer acceptance*, and *swap finalization*.

The full swap flow is depicted in the diagram below:

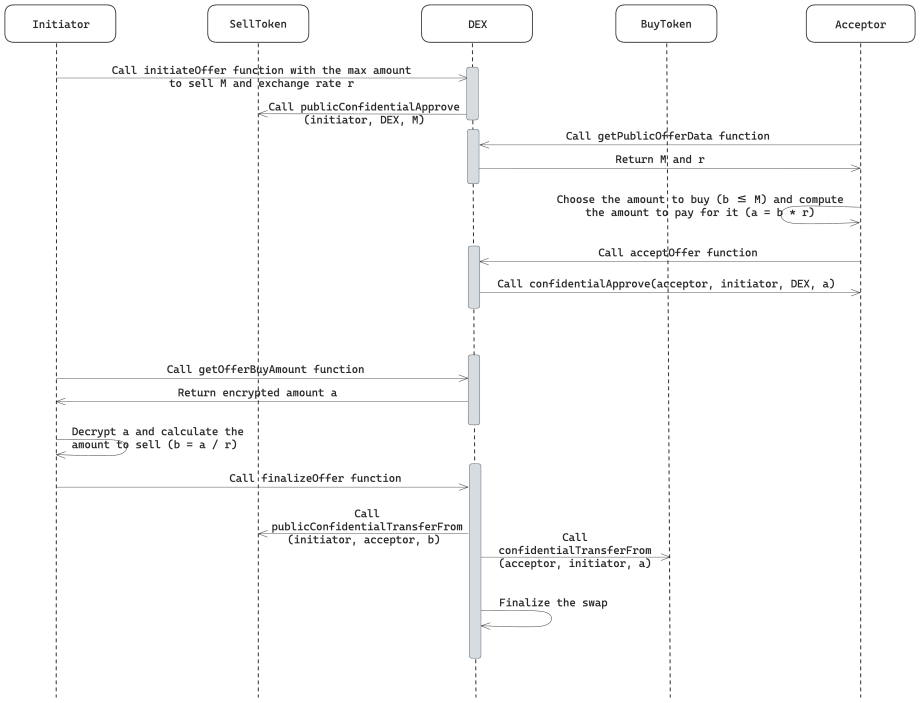

Figure 6: Confidential swap flow

3.2.1 Offer placement

The swap initiator defines and publishes the offer configuration publicly: the exchange rate r (assetBuy per assetSell) and the maximum amount M of assetSell available for exchange.

The availability of the sell token for the swap is guaranteed through the approval of M tokens to the designated ZEX contract by invoking the publicConfidentialApprove function.

3.2.2 Offer acceptance

Before accepting the offer, the counterparty must first verify that the public allowance for the specific offer exists and that the offer has not already been accepted.

The acceptor then chooses the amount b ($b \leq M$) of assetSell to receive during the swap and computes the corresponding amount of assetBuy a to pay to the offer initiator according to the agreed rate r, taking $a = br$.

After that, the acceptor must approve a assetBuy tokens via confidentialApprove, specifying the ZEX contract as an operator and the offer initiator as a spender. Also, the acceptor is required to provide a ZK proof demonstrating that the selected amount does not exceed the maximum available amount b published by the initiator.

3.2.3 Swap finalization

To finalize the swap, the offer initiator must first verify that the counterparty's confidential allowance is sufficient for the particular offer.

Afterwards, the offer initiator needs to compute the correct amount of assetSell to sell to the offer acceptor. This is done by decrypting the provided a during the confidentialApprove, and calculating the sell amount $b = \frac{a}{r}$.

The initiator then prepares the transfer of b tokens of assetSell by encrypting them with the acceptor's public key, and computing the remaining assetSell allowance $l = M - b$.

To transfer the agreed assetSell amount, the ZEX contract calls publicConfidentialTransferFrom, which adds b tokens to the acceptor's balance, while the self-encrypted allowance leftover amount l is added back to the initiator's pending balance. Finally, within the same transaction, the initiator receives the payment a in assetBuy via confidentialTransferFrom invoked by the ZEX contract by spending the acceptor's confidential allowance granted during the offer acceptance.

To summarize, users either complete the swap and exchange tokens according to the specified rate or cancel the approval at any time to stop the swapping process. At no point in time does the DEX disclose the actual amounts exchanged, publicly operating only with the exchange rate and the initial offer placement amount.

4 Solidity Smart Contracts

4.1 Confidential token state variables

4.1.1 Confidential allowance

The confidential allowance data is stored in the following struct:

```
struct ConfidentialAllowance {
    address operator;
    bytes amountEncryptionData;
    bytes amountCommitmentData;
}
```

Note that encryption and commitment data are further decoded according to the confidential token specification.

The allowances are stored in a mapping:

```
mapping(
    address approver => mapping(address spender => ConfidentialAllowance)
);
```

4.1.2 Public confidential allowance

The confidential allowance granted to the contract is stored in a mapping, where the approved amount is provided in plaintext:

```
mapping(address approver => mapping(address spender => uint256));
```

4.2 ZEX state variables

4.2.1 Offer data

The swap offer is defined by the struct below:

```
struct Offer {
    address initiator;
    address acceptor;
    address assetBuy;
    address assetSell;
    uint256 rate;
    uint256 maxAmountToSell;
    bytes amountToBuyEncryptionData;
    bytes amountToBuyCommitmentData;
}
```

Offers are stored in a simple mapping:

```
mapping(uint256 offerId => Offer);
```

4.3 Confidential token functions

4.3.1 Granting confidential allowance

To grant the confidential allowance to a registered EOA, the confidentialApprove function has to be invoked:

```
function confidentialApprove(
    address approver,
    address spender,
    address operator,
    bytes calldata amountEncryptionData,
    bytes calldata amountCommitmentData,
    bytes calldata proofData
) external;
```

The encryption and commitment data parameters store values encrypted with both approver's and spender's public keys, similarly to the transfer function from the cWETH protocol.

4.3.2 Granting public confidential allowance

To grant the allowance to the contract, the publicConfidentialApprove function is provided:

```
function publicConfidentialApprove(
    address approver,
    address spender,
    uint256 amount,
    bytes calldata newBalanceEncryptionData,
    bytes calldata amountCommitmentData,
    bytes calldata proofData
) external;
```

The encryption and commitment data parameters in this function are for storing the new balance and amount values encrypted with the approver's public key to update the balance after approval.

4.3.3 Spending confidential allowance

To spend confidential approval, the operator has to call the following function:

```
function confidentialTransferFrom(
    address approver,
    address spender,
```

```
    bytes calldata amountEncryptionData,
    bytes calldata amountCommitmentData,
    bytes calldata proofData
) external;
```

The encryption and commitment data parameters store the transfer amount encrypted with the spender's public key and the allowance amount left after transferring, encrypted with the approver's public key.

4.3.4 Spending public confidential allowance

To spend the public confidential allowance, the spender contract invokes the function below:

```
function publicConfidentialTransferFrom(
    address approver,
    address receiver,
    bytes calldata amountEncryptionData,
    bytes calldata amountCommitmentData,
    bytes calldata proofData
) external;
```

The encryption and commitment data parameters include the transfer amount encrypted with the receiver's public key, and the leftover allowance amount self-encrypted with the approver's public key.

4.3.5 Cancelling confidential allowance

To cancel the confidential allowance granted to the EOA, the approver has to use the function below:

```
function cancelConfidentialAllowance(
    address approver,
    address spender,
    bytes calldata proofData
) external;
```

In case the approver is not equal to msg.sender, the proofData contains the ZK proof to verify the possession of the approver's private key.

4.3.6 Cancelling public confidential allowance

Cancelling the public allowance granted to the contract has to be done via the following function:

```
function cancelPublicConfidentialAllowance(
    address approver,
    address spender,
```

```
    bytes calldata balanceEncryptionData,
    bytes calldata amountCommitmentData,
    bytes calldata proofData
) external;
```

The balanceEncryptionData is used to store the new DH-encrypted balance of the approver after the refund, while the amountCommitmentData represents the ElGamal commitment of the actual allowance refund amount.

4.4 ZEX functions

4.4.1 Initiating a swap offer

To create a swap offer, the initiator needs to call the following function:

```
function initiateOffer(
    address assetBuy,
    address assetSell,
    uint256 rate,
    uint256 maxAmountToSell,
    bytes calldata approveData
) external returns (uint256 offerId);
```

The approveData consists of the parameters and ZK proof required for the publicConfidentialApprove invoked within the initiateOffer function.

4.4.2 Accepting the offer

To accept the existing offer, the acceptOffer function is provided:

```
function acceptOffer(
    uint256 offerId,
    bytes calldata approveData,
    bytes calldata proofData
) external;
```

The approveData stores the data required for the confidentialApprove invoked within the acceptOffer function. It also includes the encrypted amount of assetSell to buy from the initiator. The proofData represents a ZK proof demonstrating that this amount is within the range of the defined maximum available amount of the offer.

4.4.3 Finalizing the swap

To finalize the peer-to-peer confidential swap, the offer initiator needs to call the function below:

```
function finalizeSwap(
```

```
    uint256 offerId,
    bytes calldata transferFromData
    bytes calldata proofData
) external;
```

The transferFromData consists of values required for publicConfidentialTransferFrom and confidentialTransferFrom functions invoked during the swap finalization. The proofData parameter is a ZK proof used to verify the correct decryption of the acceptor's buy amount and calculation of the initiator's sell amount.

5 ZK Circuits

All parameters outlined in this proposal are designed to be verifiable on-chain and compatible with zero-knowledge circuits.

5.1 Confidential token circuits

5.1.1 Confidential approve circuit

The list of circuit signals for the confidential approve proof is the following:

Public signals:

- Approver's public key;
- Spender's public key;
- ElGamal commitment of the approver's balance;
- ElGamal commitment of the allowance amount based on the approver's public key;
- ElGamal commitment of the allowance amount based on the spender's public key;
- New approver's balance encrypted with the approver's DH self-key;
- Random nonce used in the approver's new balance encryption;
- Allowance amount encrypted with the spender's public key-based DH shared key;
- Random nonce used during the spender's allowance amount encryption.

Private signals:

- Approver's private key;
- Approver's balance;

- Allowance amount;
- Random nonce used in the approver's ElGamal commitment;
- Random nonce used in the spender's ElGamal commitment.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided approver's public key.
2. The provided approver's balance is proven to be the one committed using the ElGamal commitment and to be greater than or equal to the allowance amount.
3. The approver's ElGamal commitment of the allowance amount was generated correctly.
4. The spender's ElGamal commitment of the allowance amount was generated correctly.
5. The approver's new balance was correctly encrypted with the approver's public key-based DH shared key.
6. The allowance amount was correctly encrypted with the spender's public key-based DH shared key.

5.1.2 Public confidential approve circuit

The list of circuit signals for the public confidential approve proof is the following:

Public signals:

- Approver's public key;
- Allowance amount;
- ElGamal commitment of the approver's balance;
- ElGamal commitment of the allowance amount based on the approver's public key;
- New approver's balance encrypted with the approver's DH self-key;
- Random nonce used in the approver's new balance encryption.

Private signals:

- Approver's private key;
- Approver's balance;

- Transfer amount;
- Random nonce used in the approver's ElGamal commitment.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided approver's public key.
2. The provided approver's balance is proven to be the one committed using the ElGamal commitment and to be greater than or equal to the allowance amount.
3. The approver's ElGamal commitment of the allowance amount was generated correctly.
4. The approver's new balance was correctly encrypted with the approver's public key-based DH shared key.

5.1.3 Confidential transferFrom circuit

The list of circuit signals for spending the confidential allowance proof is the following:

Public signals:

- Approver's public key;
- Spender's public key;
- Spender's ElGamal commitment of the allowance;
- ElGamal commitment of the amount to transfer based on the spender's public key;
- ElGamal commitment of the transfer amount based on the approver's public key;
- Transfer amount encrypted with the spender's DH self-key;
- Transfer amount encrypted with the approver's public key-based DH shared key;
- Random nonce used in the spender's transfer amount encryption;
- Random nonce used in the approver's transfer amount encryption.

Private signals:

- Spender's private key;
- Allowance amount;

- Transfer amount;
- Random nonce used in the spender's ElGamal commitment;
- Random nonce used in the approver's ElGamal commitment.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided spender's public key.
2. The provided allowance amount is proven to be the one committed using the ElGamal commitment.
3. The provided transfer amount is proven to be the one committed using the ElGamal commitment and to be greater than or equal to the allowance amount.
4. The transfer amount was correctly encrypted with the spender's public key-based DH shared key.
5. The transfer amount was correctly encrypted with the approver's public key-based DH shared key.

5.1.4 Public confidential transferFrom circuit

The list of circuit signals for spending the public confidential allowance proof is the following:

Public signals:

- Approver's public key;
- Spender's public key;
- Allowance amount;
- ElGamal commitment of the amount to transfer based on the spender's public key;
- ElGamal commitment of the allowance amount left after transferring based on the approver's public key;
- Transfer amount encrypted with the spender's public key-based DH shared key;
- Allowance amount left after transferring encrypted with the approver's DH self-key;
- Random nonce used in the spender's transfer amount encryption;

- Random nonce used in the approver's remaining allowance amount encryption.

Private signals:

- Spender's private key;
- Transfer amount;
- Random nonce used in the spender's ElGamal commitment;
- Random nonce used in the approver's ElGamal commitment.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided spender's public key.
2. The provided transfer amount is proven to be the one committed using the ElGamal commitment and to be greater than or equal to the allowance amount.
3. The allowance amount left after transferring is proven to be the one committed using the ElGamal commitment.
4. The transfer amount was correctly encrypted with the spender's public key-based DH shared key.
5. The remaining allowance amount was correctly encrypted with the approver's public key-based DH shared key.

5.1.5 Confidential allowance cancellation circuit

The list of circuit signals for cancelling the confidential allowance proof is the following:

Public signals:

- Approver's public key.

Private signals:

- Approver's private key.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided public key.

5.1.6 Public confidential allowance cancellation circuit

The list of circuit signals for cancelling the public confidential allowance proof is the following:

Public signals:

- Approver's public key;
- Allowance amount;
- ElGamal commitment of the approver's balance;
- ElGamal commitment of the refund amount based on the approver's public key;
- Approver's new balance after the refund, encrypted with the approver's DH self-key;
- Random nonce used in the approver's new balance encryption.

Private signals:

- Approver's private key;
- Approver's balance;
- Random nonce used in the approver's ElGamal commitment.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided public key.
2. The provided approver's balance is proven to be the one committed using the ElGamal commitment.
3. The provided allowance refund amount is proven to be the one committed using the ElGamal commitment.
4. The approver's new balance was correctly encrypted with the approver's public key-based DH shared key.

5.2 ZEX circuits

5.2.1 Offer acceptance circuit

The list of circuit signals for accepting the swap offer proof is the following:

Public signals:

- Acceptor's public key;

- Initiator's public key;
- Maximum amount to sell;
- Exchange rate;
- ElGamal commitment of the assetBuy amount to pay to the initiator.

Private signals:

- Acceptor's private key;
- The chosen amount to buy from the initiator;
- Random nonce used in the ElGamal commitment.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided acceptor's public key.
2. The provided amount to buy from the initiator is the one used for computing the amount to pay to the initiator committed using the ElGamal commitment.
3. The provided amount to buy from the initiator is less than or equal to the maximum amount to sell.

5.2.2 Offer finalization circuit

The list of circuit signals for finalizing the swap offer proof is the following:

Public signals:

- Acceptor's public key;
- Initiator's public key;
- Exchange rate;
- ElGamal commitment of the assetBuy amount to buy from the acceptor;
- ElGamal commitment of the assetSell amount to sell to the acceptor.

Private signals:

- Initiator's private key;
- Amount to sell to the acceptor;
- Random nonce used in the ElGamal commitment of the amount to sell.

Operating these signals, the circuit must have the following constraints:

1. The provided private key is indeed the private key of the provided initiator's public key.
2. The provided amount to sell to the acceptor is the one computed from the decrypted amount to buy from the acceptor.
3. The provided amount to sell to the acceptor is the one committed using the ElGamal commitment.

6 Security Considerations

There are several existing security challenges of the ZEX protocol that must be emphasized.

1. The initial offer allowance may be used to deduce the actual amount of tokens swapped. If the offer initiator is not careful about their public approvals, they may leak information about their previous swaps.
2. The swap offer may be griffed by accepting it with a small amount of buy tokens. One potential mitigation is to let the offer initiator specify the range of buy tokens they are comfortable accepting, but this may lead to disclosing even more information about the swap.

7 Future Work

Although this paper outlines the main functional concepts of the ZEX protocol and confidential allowances, several open directions remain for further research.

First, there is an unresolved design question regarding the management of the maximum available amount to sell during the swap offer. In the current specification, the public confidential allowance, which guarantees the availability of the sell amount, can only be spent once. The open question is the design of a more robust approach to dynamically update the public allowance amount, providing the ability to implement multi-fill offers without compromising the confidentiality of remaining allowance funds.

Second, the current solution uses multiple ZK proofs for acceptance and finalization of the offer, which introduces a significant gas overhead. Several proofs must be verified in a single transaction when performing offer manipulations. For example, during the offer finalization, one must provide a proof of the correct sell amount computation according to the provided buy amount, and two proofs for the publicConfidentialTransferFrom and confidentialTransferFrom function calls. Although such an approach may seem consistent in terms of token approval model usage separately from ZEX, it drastically increases the overall cost of the swap. One potential remedy is to delegate proof

aggregation to a trusted coordinator contract. However, this introduces new trust assumptions and expands the potential attack surface.

Third, it is possible to extend the ZEX protocol to conceal the price of the initial offer, ultimately increasing the privacy of swaps, but forcing the initiator and acceptor to haggle.

References

[1] Artem Chystiakov and Mariia Zhvanko. *Confidential Wrapped Ethereum*. 2025. URL: `https://arxiv.org/abs/2507.09231`.