Confidential ERC20 Framework using Fully Homomorphic Encryption (FHE)

Remi Gai
Inco

Kaili Wang
Circle Research

Darren Kong
Inco

Jacob Hirshman
Circle Research

Deep Gandhi
Inco

Abstract

This paper presents the Confidential ERC20 framework, which transforms ERC20 tokens into a confidential form that conceals balances and transaction amounts, with optional viewing and transfer rules to meet regulatory obligations or enhance programmatic risk management. While sender-receiver linkage remains, Confidential ERC20 balances privacy with risk management using an encryption-based approach, specifically fully homomorphic encryption (FHE), allowing operations on encrypted data without decryption.

1 Introduction

1.1 Blockchains & Privacy

Blockchain disintermediates traditional finance but falls short on privacy, as its pseudonymous design lacks the robust confidentiality needed for widespread enterprise and consumer adoption. Advanced analytics can easily trace transactions, revealing participant identities. With greater blockchain adoption, users often revert to intermediaries, like centralized exchanges, for privacy, relying on off-chain methods to meet confidentiality requirements. The re-introduction of intermediaries and custodians, however, walks back some of the disintermediated self-empowerment of the original vision while creating additional potential points of failure and cost. Privacy is vital for the financial system's integrity, protecting consumers and businesses from misuse, identity theft, and fraud. Laws often mandate privacy to secure sensitive information like payroll, vendor payments, and strategic financial data. Financial institutions enforce these protections, as data breaches can lead to costly investigations, fines, and loss of trust, potentially discouraging economic activity. Meanwhile, the repercussions on consumers for data breaches of their personal and financial information can be devastating and almost impossible to fully remediate, particularly in a blockchain context where wallet information can expose historical transactions far beyond any immediately exploited transaction, which could include payments to healthcare providers or involving human rights actors and political dissidents.

1.2 Confidentiality vs. Anonymity

Privacy is not a one-dimensional feature. There are primarily two aspects of a payment that could be private:

1. *who* (sender and receiver)
2. *how much* (the amount transferred)

Anonymity is achieved when the sender/receiver information is private. Confidentiality is achieved when the amount in the transaction is private.

	Anonymity	**Confidentiality**
Public information	? sent $10 to ?	Alice sent ? to Bob
Scenarios when needed	Paying your doctor or donating to a political campaign	Payroll, supply chain/vendor payments
Examples	Tornado Cash	Confidential ERC20s

Table 1: Anonymity vs. Confidentiality

Though anonymity typically provides stronger privacy assumptions than confidentiality, it also can pose threats from a risk management standpoint, making it easier for bad actors to obfuscate illegal activity. Conversely, confidentiality is sufficient for many use cases in the financial system: employee payroll, supply chain vendor payments (where payment amount should not be revealed to vendor competitors), or even P2P payments (ex. Venmo).

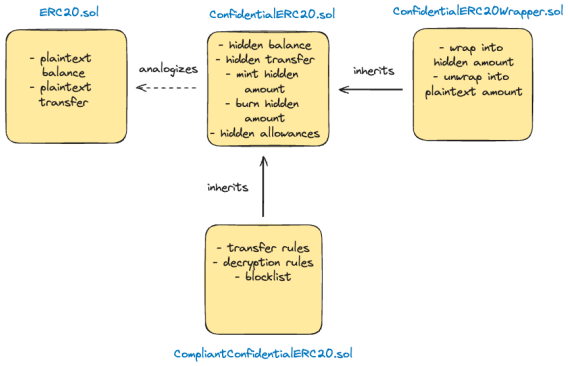
On-chain confidentiality is essential for mass adoption, unlocking new assets, applications, and audiences for blockchain technology. Given this, Confidential ERC20s, as discussed in this paper, aim to purely achieve confidentiality to offer a permissionless, programmable, and composable on-chain primitive tailored for EVM dApps.

2 Confidential ERC20 Framework

2.1 ERC20 Standard

The ERC20 standard [1] is a foundational technical specification for creating fungible tokens on the Ethereum blockchain. Introduced in 2015, ERC20 outlines a set of standardized rules and functions that tokens must implement to ensure compatibility within the EVM ecosystem. This standard has become a cornerstone of Ethereum's ecosystem, enabling a wide range of tokens with varied use cases—from utility tokens that power protocols and dApps, governance tokens for DAOs and protocol decision-making, and stablecoins pegged to traditional assets to community-driven tokens like meme coins and other socially-oriented tokens.

2.2 Confidential ERC20



The confidential ERC20 framework preserves all core functionalities of the standard ERC20 while adding features to ensure confidentiality. Designed natively, it enables confidential minting and burning of tokens. Additionally, it can serve as a wrapper to provide these confidentiality features to existing ERC20 tokens via

wrapping and unwrapping mechanisms. Transfer and decryption rules can also be customized for compliance, allowing flexibility in regulatory contexts.

Key features:

- **Encrypted balances:** User balances are encrypted, ensuring that only authorized parties, such as the users themselves or designated regulators, can view them.
- **Encrypted transfers:** When transferring wrapped assets, transfer amounts between addresses are encrypted so that the details of the transactions are not exposed to the public.
- **Delegated viewing (optional):** Delegation of view permissions to specific parties, such as auditors or regulators, without compromising the privacy of token holders.
- **Programmable transfer rules (optional):** Transfer rules can be programmed at the smart contract level to enforce compliance rules such as anti-money laundering (AML) blacklisting or transaction limits.

3 Fully Homomorphic Encryption (FHE)

The Confidential ERC20 token framework leverages Fully Homomorphic Encryption (FHE) to achieve confidentiality and compliance within an evolving regulatory landscape. We explore FHE's application within the Ethereum Virtual Machine (EVM) and its role in this framework. While FHE serves as the primary privacy-enhancing technology (PET) for this study, the framework's interfaces and features could be adapted to integrate other encryption-based technologies, such as Trusted Execution Environments (TEE), garbled circuits (GC), Multi-Party Computation (MPC), or similar alternatives.

3.1 What is FHE?

Fully Homomorphic Encryption (FHE) 2 enables computations on encrypted data without decryption. Its "homomorphic" property allows arithmetic operations on ciphertexts that, upon decryption, yield results identical to those of operations performed on the original plaintext data. FHE builds on public-key encryption (PKE), using a pair of keys: a public key for encryption and a private key for decryption.
Encryption: The public key encrypts plaintext m into ciphertext ct, expressed as:

$$\text{Enc}(pk, m) \longrightarrow ct$$

Decryption: The private key decrypts ciphertext ct back to plaintext m, expressed as:

$$\text{Dec}(sk, ct) \longrightarrow m$$

3.2 FHE schemes

While numerous Fully Homomorphic Encryption (FHE) schemes such as TFHE (Torus FHE), CKKS (Cheon-Kim-Kim-Song), and BFV (Brakerski-Fan-Vercauteren) exist, we leverage the TFHE scheme 3 for its suitability in the context of smart contracts due to the following key properties:

- **Broad Operation Support:** TFHE enables a wide range of homomorphic operations, providing maximum flexibility to accommodate diverse use cases across various industry verticals.
- **Exact Computation:** Accuracy is critical in scenarios such as decentralized finance (DeFi). TFHE guarantees exact ciphertext computations, avoiding approximations that could compromise the integrity of results.

- **Non-Leveled Structure:** Unlike leveled schemes, TFHE does not impose restrictions on the number of operations, making it ideal for smart contracts where the complexity and number of FHE operations can vary significantly.

Our initial prototype uses the TFHE-rs implementation from Zama. 4

3.3 Trust Assumptions

Decryption: Ciphertexts are encrypted under a global FHE public key for composability, with decryption done by a trusted authority that holds the private key. If a single private key is used, it remains with one authority; if managed through Multi-Party Computation (MPC), multiple authorities share the key. For example, using an n-of-m threshold, the private key can be split among m parties, with n parties required to sign for decryption. To enhance the security of the decryption system, various techniques—such as Distributed Key Generation (DKG), key rotation, storing key shares within Trusted Execution Environments (TEE), and collaborating with reputable entities—can be employed to mitigate risks from hacks and collusion.

Computational integrity: FHE computation is deterministic, and in the blockchain context, we may leverage techniques like Zero-Knowledge Proofs (ZKP), fraud proofs, and/or consensus protocols to agree on computational validity.

- **ZKP and computational correctness:** Using ZKP to introduce verifiable proofs of the correctness of FHE computation is a promising avenue of research but is too computationally intensive 5 to be practical in the near term.
- **Fraud proofs in optimistic systems:** Fraud proofs in optimistic systems (e.g., roll-ups) support efficient off-chain computation but require a finality period of at least seven days. This delay poses challenges in balancing security and user experience, as sensitive encrypted information may be unintentionally revealed or maliciously decrypted during the challenge-response phase, risking data leakage or financial loss (e.g., unwrapping balances).
- **Consensus protocols:** Consensus protocols, on the other hand, rely on replicating computation among distributed participants, which can be resource-intensive but, in exchange, offer verifiability on the computational integrity and provide instant finality guarantees to the decryption network.

3.4 Applying FHE to the EVM

Smart contract execution environments like the Ethereum Virtual Machine (EVM) can be augmented to allow FHE-encrypted types and computation. In doing so, EVM developers can build a new category of smart contracts where the on-chain states can also be private.

Key smart contract features of an FHE-augmented EVM:

- **Global Network Key:** All transaction inputs and on-chain states are encrypted under a global network key, ensuring state confidentiality and composability across smart contracts.
- **New encrypted data types:** This includes encrypted boolean (*ebool*), address (*eaddress*), and various unsigned integers (*euint(8,16,32,64,128,256)*).
- **Operations on encrypted types:** Supported operations include:
  - Arithmetic functions such as addition, subtraction, multiplication, and selective division or remainder operations.
  - Bitwise operations like AND, OR, XOR, as well as bit shifts (left and right).
  - Comparison operators like equality, inequality, greater than, and less than checks.
  - Additional functions like minimum, maximum, negation, and logical NOT.

  - Unique FHE-specific operators such as CMUX/SELECT for conditional assignments, and PNRG for generating random encrypted integers directly on-chain.

- **Integration into the EVM:** FHE can be integrated into the EVM at the precompile level or as a co-processor to existing chains like Ethereum and Arbitrum, enabling confidential tokens (e.g., *cUSDC*) to exist across ecosystems.

- **Co-processing approach:** This approach involves external monitoring of the origin chain and executing the FHE instructions off-chain. Any decryptions happen asynchronously, and the decrypted value is settled back on-chain via a callback function.

3.5 Why FHE

Confidential transfers can use either commitment-based or encryption-based methods. While commitment-based approaches are viable, encryption-based models offer distinct advantages in composability and flexible decryption access control.

Commitment-based methods rely on client-side off-chain secret storage of the committed data. This poses challenges for multi-party coordination and limiting composability across dApps due to application-specific circuits and off-chain encryption keys. In contrast, FHE systems support on-chain ciphertext storage, enabling smart contract-level computational logic over encrypted data and enhancing composability over private states through permissionless user interactions.

Under a commitment-based model, users must define a viewing key, making access control such as delegating or revoking rigid. FHE, however, uses a shared encryption key, enabling a designated decryption authority to decrypt on behalf of users. This approach makes decryption rules more flexible and upgradable based on on-chain rules and governance.

Commitment-based models are better for high-anonymity needs, such as medical privacy. Encryption-based models suit use cases where confidentiality is needed without full identity masking, like in supply chain transactions, in which the parties are typically known. It's worth noting that FHE remains computationally intensive and, just like ZK, will require hardware acceleration. Nonetheless, FHE is practical for simpler EVM use cases and is only used to compute hidden state transitions.

For a more detailed comparison table, see the Appendix.

4 Implementation

New functions in ConfidentialERC20Wrapper:

Wrapping and unwrapping:
When a user initiates the wrapping function, the original ERC20 tokens are deposited into the wrapper contract, and an equivalent amount of confidential ERC20 (*cERC20*) tokens is minted to the user's address. Note that this wrapping step leaks information about the deposited amount.

```
function wrap(uint64 amount) external
```

Conversely, the unwrapping process allows users to burn their *cERC20* tokens in exchange for an equivalent amount of the original ERC20 tokens. This process also reveals the converted amount back to plain text.

```
function unwrap(uint64 amount) external
```

New functions in ConfidentialERC20:

User Balance:
A user balance is represented by a mapping between the user address (externally owned account or smart contract wallet) and *euint64*. The *euint64* data type represents an encrypted 64-bit unsigned integer. The *balanceOf* function returns the ciphertext representing the encrypted balance of the requested account, and the decryption network will verify the access control rules and decrypt it on behalf of the user.

```
mapping(address => euint64) public balances;

function balanceOf(address account) external returns (euint64)
```

Transfer Amount:
The transfer amount in this framework is represented as an encrypted value, such as *euint64*. In FHE, conditional logic (like if/else statements) cannot be directly applied to check balances, as any revert action would reveal information about the underlying plaintext value. Instead, the encrypted amount is compared homomorphically against the sender's balance using *TFHE.le* to verify if it is less than or equal to the balance, which returns an encrypted boolean (*ebool*). The *ebool* is then passed into a conditional multiplexer operation (*TFHE.select*), which returns the hidden transfer amount if the boolean is true, or an encrypted zero if false. The sender's and recipient's balances are then adjusted based on the result, where the transfer amount reflects either the actual value or zero, depending on the comparison outcome. To verify the accuracy of the newly transferred balance, the recipient can request the authority to decrypt their balance and confirm the correctness of the amount[1].

```
function transfer(
    address to,
    euint64 amount
) external {
    ebool canTransfer = TFHE.le(amount, balances[msg.sender]);
    euint64 transferValue = TFHE.select(canTransfer, amount, TFHE.asEuint64(0));
    balances[to] = TFHE.add(balances[to], transferValue);
    balances[from] = TFHE.sub(balances[msg.sender], transferValue);
}
```

Approval / Allowance:
The *allowance* function returns the remaining tokens a spender is permitted to spend on behalf of the token owner, with the value stored and returned in an encrypted format (*euint64*). The *approve* function authorizes the spender to transfer a specified amount of tokens on behalf of the token holder, also handling the amount in encrypted form.

```
function allowance(address spender) external view override returns (euint64)

function approve (
    address spender,
    euint64 amount
) external returns (bool)

function increaseAllowance(
    address spender,
    euint64 amount
) external returns (ebool)

function decreaseAllowance(
    address spender,
    euint64 amount
) external returns (ebool)
```

5 Benefits of Confidential ERC20

Traceability of transfers

Confidential ERC20 benefits from the transparent nature of blockchains, e.g., being able to monitor the flow of funds while preserving the confidentiality of the transfer amounts and user balances. On a block explorer like Etherscan, transfers would be made public between the sender and recipient, with the amount transferred hidden.

[1] Alternatively, a commitment-based viewing key approach, generated on the client side, can be introduced, or an encryption-based method, where the ciphertext is re-encrypted using a specified viewing key, can be employed.

Programmable Decryption Rules

Decryption rules can be programmed at the smart contract level, and the ability to view/decrypt a hidden balance can be delegated to specific entities such as regulators, government bodies, auditors, or the original token issuer to fulfill their roles in monitoring compliance and conducting audits.

Examples of decryption rules:

Role	Permissions	Purpose
User	Full access	Manage personal transactions
Friends & Family	Limited access by user	Assist user
Asset Issuer	Issued asset data	Monitor asset lifecycle
Government Agency	Legal-based	Compliance monitoring
Auditor	Conditional access	Audit transactions

Table 2: Roles and Permissions

Example of decryption based on authority:

For example, the viewing key can be delegated to a specific authority, which can whitelist itself using `TFHE.allow` to gain the right to decrypt and view the hidden balance of a particular address. In this scenario, the authority is the smart contract owner. Still, it could also be programmed through governance to be a specific externally owned account (EOA) owned by a third party, such as Chainalysis.

```
function adminViewUserBalance(
        address user
    ) public onlyOwner
{
  TFHE.allow(balances[user], owner()); // explicitly allowing a user to decrypt the hidden
    balance of a user
}
```

It's important to note that viewing access to a ciphertext can be revoked for specific addresses based on on-chain access rules. However, this may have limited effectiveness, as any decrypted information should be treated as potentially leaked; for instance, a viewer could retain a copy, making the revocation ineffective in fully securing the data.

Programmable Transfer Rules

Programmable compliance enables enforcement of transfer rules based on factors such as user identity, or blacklist status. For example, a Decentralized Identifier (DID) could store encrypted attributes like accredited investor status, which is then used to determine transfer eligibility. Since FHE prevents observable transaction failures by using a conditional multiplexer (`TFHE.select`), a transfer that doesn't meet compliance conditions would proceed with an encrypted zero amount[2], concealing the failed condition from external observers while maintaining confidentiality.

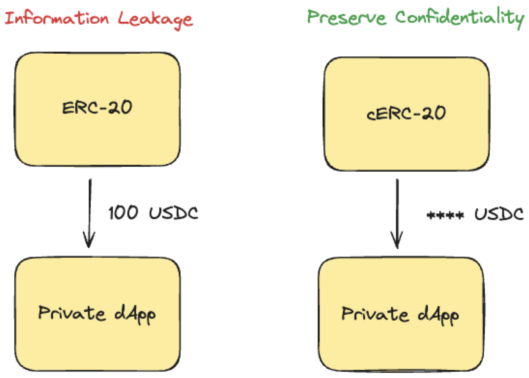
Granular control over transfer rules is especially valuable for tokenized asset classes subject to strict regulatory requirements. For a more comprehensive framework, the ERC-3643 standard, also known as T-REX (Token for Regulated EXchanges) 6, is designed to enable the issuance, management, and transfer of security tokens in compliance with regulations. It allows for a more granular design and can be combined with other OnchainID 7 standards, such as ERC734 and ERC735.

[2]See Appendix for risks that come with allowing zero transfers on the Confidential ERC20.

6 Extensibility and Composability

Why Choose Confidential ERC-20s

Privacy can be implemented at various levels: as a standalone public or permissioned blockchain, a decentralized application/protocol, or a foundational primitive. Each approach has unique advantages and trade-offs. We focus on the confidential ERC-20 framework, a highly modular and granular component that serves as a foundational building block, maximizing composability and extensibility for seamless integration with other privacy-preserving applications across an ecosystem.



Just as the ERC-20 standard became a foundational layer for decentralized applications like Uniswap (automated market maker) and Compound (money market), the confidential ERC-20 standard provides a core primitive for building privacy-preserving applications.

This framework's ability to support composability over encrypted data makes it exceptionally extensible, enabling applications to unlock new possibilities while preserving privacy. For instance, users can deposit an encrypted amount of confidential ERC-20 tokens directly into a private trading or lending platform, eliminating the need to transfer a plaintext amount and subsequently wrap it within the application—an approach that would otherwise reveal sensitive information.

Additional Use Cases

Here are additional examples of use cases that confidential ERC20 can enable and extend to:

Payment-related use cases

- **Private Cross-border payments:** Enable private international transactions, protecting sensitive financial details while complying with local regulations.
- **Private Token vesting:** Manage and automate the distribution of vested tokens, ensuring confidentiality of allocation schedules and amounts.
- **Private B2B payroll:** Facilitate private payroll between businesses, safeguarding salary details and ensuring compliance with regulatory requirements.
- **International Remittances:** Provide a secure and private channel for transferring funds across borders, protecting senders and recipients from exposure and fraud.

Defi-related use cases

- **Private AMM & Darkpools:** Execute private token swaps, where transaction amounts are encrypted, preventing frontrunning and protecting trading strategies. The implementation presents interesting challenges that can be further explored in future research.
- **Private RWA Tokenization:** Tokenize real-world assets with privacy-preserving features, safeguarding sensitive ownership and transaction details.
- **Blind auction:** Facilitate auctions where the bid remains confidential until the auction concludes, ensuring a fair and competitive bidding process.
- **Private lending:** Enable private loans with confidential terms and collateral, preserving borrower privacy. Potentially support undercollateralized lending by incorporating on-chain credit scores to assess creditworthiness privately.

7 Conclusion

We introduced the Confidential ERC20 framework as a programmable confidentiality primitive for asset issuers, application developers, and end users. Issuers and developers can customize features like compliance rules (decryption and transfer), operational workflows, and other application-specific requirements while maintaining confidentiality throughout the process. On-chain composability allows for seamless extensibility, enabling new dApps to emerge and be built on this foundational framework. This primitive combines the familiar user experiences and financial data privacy of Web2 with the trustlessness and security of blockchain systems.

8 Acknowledgements

We thank Dan Boneh and Michael Mosier for their contributions to this research.

Works Cited

1. F. Vogelsteller and V. Buterin, "ERC-20: Token Standard," *Ethereum Improvement Proposals*, Nov. 19, 2015. [Online]. Available: `https://eips.ethereum.org/EIPS/eip-20`. Accessed: Sep. 26, 2024.

2. C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford University, Stanford, CA, 2009. [Online]. Available: `https://crypto.stanford.edu/craig/craig-thesis.pdf`. Accessed: Sep. 26, 2024.

3. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," *Cryptology ePrint Archive*, 2016. [Online]. Available: `https://eprint.iacr.org/2016/870`. Accessed: Sep. 26, 2024.

4. I. Chillotti, M. Joye, and P. Paillier, "Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks," *Cryptology ePrint Archive*, 2021. [Online]. Available: `https://eprint.iacr.org/2021/091.pdf`. Accessed: Sep. 26, 2024.

5. L. T. Thibault and M. Walter, "Towards verifiable FHE in practice: Proving correct execution of TFHE's bootstrapping using plonky2," *Cryptology ePrint Archive*, Mar. 15, 2024. [Online]. Available: `https://eprint.iacr.org/2024/451`. Accessed: Sep. 26, 2024.

6. J. Lebrun, *et al.*, "ERC 3643 white paper," *Token for Regulated Exchanges White Paper*, 2024. [Online]. Available: `https://github.com/TokenySolutions/T-REX/blob/main/docs/TREX-WhitePaper.pdf`. Accessed: Sep. 26, 2024.

7. Tokeny Sàrl, "onchain-id/solidity: Smart contracts for secure blockchain identities, implementation of the ERC734 and ERC735 proposal standards," *GitHub*, 2024. [Online]. Available: `https://github.com/onchain-id/solidity`. Accessed: Sep. 26, 2024.

8. S. Nakamoto, "A peer-to-peer electronic cash system," *Bitcoin.org*, 2008. [Online]. Available: `https://bitcoin.org/bitcoin.pdf`. Accessed: Sep. 26, 2024.

9. J. Burleson, *et al.*, "ZKPs and regulatory-compliant privacy," *a16z crypto*, 2022. [Online]. Available: `https://api.a16zcrypto.com/wp-content/uploads/2022/11/ZKPs-and-Regulatory-Compliant-Privacy.pdf`. Accessed: Sep. 26, 2024.

10. V. Buterin, J. Illum, M. Nadler, F. Schär, and A. Soleimani, "Blockchain privacy and regulatory compliance: Towards a practical equilibrium," Sep. 9, 2023. [Online]. Available: `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4563364`. Accessed: Sep. 26, 2024.

11. J. Beal and B. Fisch, "Derecho: Privacy pools with proof-carrying disclosures," *Cryptology ePrint Archive*, Feb. 23, 2023. [Online]. Available: `https://eprint.iacr.org/2023/273`. Accessed: Sep. 26, 2024.

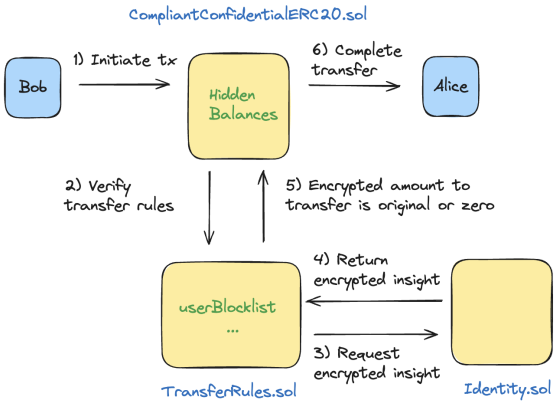
12. S. van Schaik, *et al.*, "SGX.Fail: How stuff gets eXposed," *SGX.Fail*, Nov. 30, 2022. [Online]. Available: `https://sgx.fail/files/sgx.fail.pdf`. Accessed: Oct. 9, 2024.

Appendix

Transfer Rules

Example: transfer rules based on transfer limit, age, and blacklist: Payment dApps can enforce transfer compliance rules directly within the smart contract. We use a DID registry smart contract, where the user's age is encrypted and stored on-chain. This allows token transfers to be programmatically controlled based on encrypted identity attributes, ensuring compliance is enforced at the contract level rather than off-chain. Let's take a scenario in which transfers can be authorized based on a combination of transparent and encrypted conditions:

- **Condition 1 (plaintext):** Verify that the addresses involved are not blacklisted.
- **Condition 2 (hidden):** Ensure the transfer amount is below the limit (20,000 tokens).
- **Condition 3 (hidden):** Ensure the user is over 18 years old.



Blacklist conditions can be evaluated using plaintext logic, while multiple confidential conditions are combined through homomorphic logic gates like `TFHE.or`, `TFHE.not`, and `TFHE.and`. These operations return

an encrypted boolean (*ebool*) that indicates whether all specified conditions are met, preserving privacy throughout the evaluation process.

Potential Vector of Attack with Zero Transfers

A malicious actor may transfer an encrypted zero amount to a large number of accounts to create spam and attempt to obfuscate transaction paths. However, such an attack would be expensive; zero transfer transactions must pay transaction gas and FHE compute fees.

Moreover, because the flow of funds remains traceable within the confidential ERC20 framework, a designated authority, if programmed, could ultimately investigate each recipient individually to trace the funds.

To mitigate such spam, zero-value transfers can be proactively blocked by homomorphically comparing the encrypted transfer amount to zero using equality comparisons (`TFHE.eq`). The resulting encrypted boolean (*ebool*) is then decrypted via a callback, determining whether the transfer proceeds. While this approach effectively filters out zero-value transfers, it may introduce processing delays, potentially affecting the user experience.

Related Work

Blockchain proponents have been strongly associated with privacy from the early days. The Bitcoin whitepaper 8 directly mentions privacy, and early private cryptocurrencies like Zcash and Monero demonstrate market demand. The blockchain industry has been a test bed for new, promising cryptographic technologies utilized for privacy, including ZK, TEE, MPC, and FHE. Each technology has different attributes, providing varied approaches and empowering a range of use cases. Many of these technologies are being tested on smart contract environments like the EVM to extend privacy to existing users and applications.

Zero-Knowledge Proofs (ZK)

Zero-knowledge proofs (ZKPs) are cryptographic methods that allow a party (prover) to convince another party (verifier) that a statement is true without revealing details about the statement itself. ZK proofs are also succinct (compact) in relation to the statements being proven. ZKPs can be successfully utilized purely for their verifiable succinctness in transparent use cases like ZKEVMs. However, privacy projects, like Zcash and Tornado Cash, utilize ZKPs to enable transactions in which the sender, recipient, and amount are hidden from public view (proven off-chain client-side) — while maintaining the verifiability of the transaction's validity.

ZK and Compliance

Existing ZK compliance tools are often built to provide selective disclosure of funds. Tornado Cash provides "deposit notes," which can be later used to generate a report tracing the funds back to the original deposit address. Zcash, a non-EVM zk-based privacy blockchain, also allows users to generate a viewing key, derived from the user's private spend key, that enables viewing the transaction details of a specific shielded address. The report and viewing key allow users to selectively reveal the source of funds to third parties like regulators or exchanges, providing a way to prove the legitimacy of funds.

Compliance Landscape

In August 2022, the U.S. Treasury's OFAC sanctioned Tornado Cash, citing its role in laundering over $7 billion of illicit funds, including cryptocurrency stolen by North Korean hacking groups. The sanctions prohibit U.S. persons and entities from interacting with Tornado Cash or its associated addresses. The sanctions are currently being challenged in the U.S. judicial system.

Proof of Innocence

New ZK-based privacy protocols have sprung up in reaction to Tornado Cash's sanctions, iterating on its' design and seeking to improve compliance. a16z Crypto Research, among others, listed several possible modifications 9 to Tornado Cash (Burleson et. al). "Proof of Innocence," protocols like Privacy Pools by Buterin et. al 10 introduce membership proofs, which prove that a withdrawal is linked to one of several deposits, and exclusion proofs, which show that a withdrawal does not originate from specific deposits. These proofs prove "innocence," without revealing exactly which withdrawal a deposit is linked to. Derecho (Beal, Fisch) 11 takes this one step further, showing how chained proofs of innocence can be generated when users transact within a privacy pool. While proofs of innocence are useful, it is unclear whether these proofs would satisfy regulators.

Regulatory Update

As there are ongoing updates in the regulatory landscape, there may also be an ongoing need to update the underlying systems to integrate with changes to the regulatory framework. ZK-based privacy systems often have more rigidity when updating a shielded pool with new logic and mechanisms. Thus, these systems are innovative but may encounter UX (e.g., user-based proof generation times) and DevEx challenges that are actively being worked on.

Encryption versus Commitment-based approaches

We compare the trade-offs between commitment-based and encryption-based approaches for confidential payments. Typically, a commitment-based approach leverages zero-knowledge (ZK) proofs, while an encryption-based approach can employ technologies like Fully Homomorphic Encryption (FHE), Multi-Party Computation (MPC), Garbled Circuits (GC), and Trusted Execution Environments (TEE)

Properties	**Commitment-based**	**Encryption-based**
Examples	Zether, CAPE, Railgun	Confidential ERC20 over FHE
Compute overhead	State transition proofs are generated on the client side, distributing the computational load across users and scaling with the user base.	State transitions occur at the blockchain level, requiring more intensive scaling solutions to handle large transaction throughput.
Privacy	Each user maintains a unique private/public encryption key on the client side. Encryption and decryption are performed entirely on the client side, keeping private keys and sensitive information exclusively with the user.	Encryption often relies on a shared private key managed by one or more authorities (via MPC), which introduces potential risks of collusion among authorities.
Decryption Liveness	Each user manages their own private decryption key independently, without relying on any third party.	Liveness relies on the availability of the decryption authority to perform decryptions.

Decryption Rules	Rigid: Each confidential transfer includes a proof specifying the recipient's viewing key. Changing the viewing key for historical encrypted data, such as delegating access to an auditor, is not feasible since the viewing key is set at the time of proof generation.	Flexible: The decryption authority follows decryption rules specified on-chain, which can be dynamically updated by the smart contract owner or through governance.
Key Rotation	Difficult: Fixed viewing keys during proof generation complicate key rotation.	Achievable through re-encryption or homomorphic key rotation over FHE.
Loss of Viewing Key	Problematic: Data is encrypted under a specific viewing key during proof generation and cannot be changed retroactively.	Flexible: Viewing rules can be changed on-chain, enabling the delegation of decryption to different users.
Decryption Revocation	Not feasible within the ZK framework.	Key revocation is feasible but limited in effectiveness since any previously decrypted information should be considered compromised.
Concurrency/Front-running Attack [3]	State transition proofs must be generated, which may lead to state conflicts if verification occurs over an outdated state.	Encrypted state can be transformed directly.
Composability over Private States	Limited: ZK proofs cannot perform state transitions directly on encrypted data. Multi-party operations without information leakage require additional encryption methods, such as MPC.	Highly composable: All ciphertexts are encrypted under a shared public key, enabling seamless computations across encrypted data and facilitating private dApp interaction.
Proof Generation (Client side)	Off-chain proof generation ensures that newly encrypted data is correctly formed, including data integrity checks (e.g., sufficient balances) and adherence to circuit logic (e.g., transfer rules).	Off-chain proof generation confirms the integrity of encrypted data and the user's knowledge of the plaintext, preventing reuse of identical ciphertext as input.

Other Technologies

Emerging projects are using and developing TEE, MPC, and FHE-based systems for privacy. Trusted Execution Environments, TEEs, are specialized hardware designed and fabricated by manufacturers like Intel, Nvidia, and others to provide secure and isolated environments for sensitive applications. TEEs are versatile and performant but have trust assumptions on the manufacturer, supply chain, data center, and operator as they may be susceptible to various physical attacks 12. Secure Multi-Party Computation, MPC, are cryptographic methods allowing several parties to compute a function using private inputs jointly. To date, they have found traction for off-chain infrastructure/tooling use cases like asset custody and embedded wallets. These systems and technologies are ongoing areas of research and development.

[3]An attacker can disrupt a user by front-running transactions. For instance, if Alice generates a proof for her current encrypted balance and another user, Bob, transfers tokens to her before her transaction is processed, her proof becomes outdated, causing her transaction to be rejected. Normally, Alice could update her proof and resubmit, but if an attacker repeatedly floods the network with transfers to Alice's account, it could render her account effectively unusable due to constant state changes. It's worth noting that there are techniques to solve this problem by having a "pending queue" for incoming balances, or separating a balance into "pending" and "available", and "rolling up" to the account balances at a later point.