Proof of concept for a Ethereum Virtual Machine on Cryptonote*

Allard Dijk
Lead Developer
BitBender
The Netherlands
allard@bitbender.nl

Dominique Schröder
Friedrich-Alexander University
Erlangen-Nürnberg
Germany
dominique.schroeder@fau.de

May 5, 2023

Abstract

The focus of this paper is to present a proof of concept that investigates the feasibility of integrating the code of a program directly into the blockchain, using the Beldex blockchain — a Monero fork that prioritizes privacy in blockchain applications. The focus of this work is to present a proof of concept that investigates the feasibility of integrating the code of a program directly into the blockchain. The results show that it is possible to achieve this goal and that it has the potential to enhance the security and efficiency of smart contracts. In addition, future work will concentrate on exploring the privacy implications of integrating smart contracts into the Beldex blockchain.

1 Introduction

Cryptocurrencies like Bitcoin and Monero eliminate the need for a central authority and offer a publicly verifiable, decentralized system. These currencies consist of a public ledger (such as a blockchain) for publishing transactions and a transaction scheme for specifying transaction format and validity. For instance, a simple transaction in Bitcoin involves a signed statement indicating that "Pseudonym i pays amount a to pseudonym j". While the public nature of cryptocurrencies facilitates easy verification of the ledger, it also poses a privacy risk for users as different pseudonyms associated with the same user can be linked through simple or sophisticated methods by observing transactions on the public blockchain [And+13; Bar+12; KKM14; Mei+13; MO15; RH13; SMZ14]. This makes transactions traceable, and users can be re-identified.

To address the privacy issue, various privacy-enhancing technologies have been proposed by the academic community and cryptocurrency community [Bon+14; BS+14; GK15; Hei+17; Kos+16a; Max13; Max15; Mie+13; Poe+17; RMS17; RMSK17; Sab13; VR15; Zie+15]. This has resulted in the emergence of multiple privacy-focused cryptocurrencies [Mon14; Zca16; Zco16]. Monero [Mon14], with a market capitalization of $3 billion [Mon23], is the largest of these privacy-focused cryptocurrencies. Unlike other approaches like Zerocash [BS+14], which requires a trusted cryptographic setup to achieve anonymity, Monero's main design goal is to avoid any form of trusted setup, making it closer to the decentralized spirit of cryptocurrencies.

*Thanks to Beldex for funding this research

Most cryptocurrencies, such as Bitcoin and Monero, do not have a full programming language built into their systems or some functions have been disabled. They typically only provide support for simple transactions, with limited flexibility and functionality. Ethereum, however, is an exception as it has a full-fledged programming language built into its blockchain, allowing developers to build and run complex decentralized applications and smart contracts. This provides a higher level of functionality and versatility compared to other cryptocurrencies.

Smart contracts and privacy-preserving cryptocurrencies can appear to be conflicting concepts. On one hand, smart contracts require access to the variables and values in order to execute, meaning they need to "read" this information. On the other hand, privacy-preserving cryptocurrencies aim to keep the values and transactions private and secure. This creates a contradiction, as the need for smart contracts to access the information seems to undermine the privacy of the values. In other words, the very nature of smart contracts, which require transparency and accessibility, seems to contradict the principles of privacy-preserving cryptocurrencies. This presents a challenge for the development of privacy-preserving smart contracts.

1.1 Beldex

Beldex is a cryptocurrency that was forked from Monero with a primary focus on privacy. Its goal is to create a comprehensive platform that offers full privacy for its users by anonymizing transactions, communications, and online activities. To achieve this, Beldex has designed an ecosystem that incentivizes a network of validators and operates on a sustainable economic model. The Beldex ecosystem comprises of several components that work together to deliver a seamless and private user experience. These components include the Beldex blockchain, an anonymous decentralized messenger, a privacy-focused browser, a secure wallet, and the Beldex Bridge, which links the Beldex ecosystem to other blockchain networks. The platform is designed to provide a highly secure and private environment for users, enabling them to transact and communicate in complete anonymity. With its innovative and privacy-focused approach, Beldex aims to set a new standard in the cryptocurrency industry, offering a truly private and secure solution for users.

1.2 The Goal of this Proof of Concept

The purpose of this proof of concept is not to provide a complete solution for privacy-preserving smart contracts. Instead, its focus is to demonstrate the ability to embed code within a transaction on the Beldex blockchain. This proof of concept serves as a stepping stone for further development and exploration of privacy-enhancing technologies within the blockchain space. It highlights the potential for adding new functionality and capabilities to the Beldex blockchain, while also showcasing its commitment to privacy. By demonstrating the feasibility of embedding code in a transaction, this proof of concept opens up new possibilities for the Beldex blockchain and its potential use cases.

2 The Model

As blockchain technology continues to evolve, the potential for smart contracts to revolutionize various industries is becoming increasingly apparent. However, with the power to execute complex automated transactions comes the challenge of ensuring privacy and security for all parties involved. This is where the vision for privacy-preserving smart contracts comes in — by developing a system

that allows for the execution of smart contracts while maintaining privacy, the goal of the Beldex ecosystem is creating an efficient, scalable, and secure platform for decentralized applications. Of course, realizing this vision requires overcoming numerous research challenges.

In this section, we will provide a detailed description of the underlying model for our proof of concept. We will explain the different components that make up our system and how they interact with each other. We will also discuss the open problems that we have encountered during the development of our proof of concept, including issues related to scalability, privacy, and security. Finally, we will explain the limitations of our proof of concept, including the fact that it is still in the early stages of development and may not be suitable for use in production environments. Nonetheless, our work represents an important step forward in developing privacy-preserving smart contracts, and it has the potential to contribute significantly to the ongoing efforts to create a more secure and efficient blockchain ecosystem.

2.1 UTXO vs. Account-based Model

In the context of cryptocurrencies, the blockchain can be viewed as a public ledger that records all transactions that have occurred on the network. However, it is also possible to think of the blockchain as a record of state transitions. Each block in the blockchain represents a particular state of the network, and the process of adding a new block involves transitioning from one state to another. This is accomplished by validating and verifying transactions, updating the ledger's state, and then appending a new block to the chain. This process ensures that the blockchain remains an accurate and up-to-date record of all transactions that have occurred on the network.

There are mainly two competing models for expressing state transitions in blockchain-based systems: the UTXO model and the account-based model. The UTXO model, which is used in Bitcoin and other similar cryptocurrencies, records the current state of the network as a collection of unspent transaction outputs. In this model, each output represents a specific amount of cryptocurrency that its owner can spend in a new transaction. The account-based model, which is used in Ethereum and other similar cryptocurrencies, records the current state of the network as a collection of account balances. In this model, each account has a balance that can be increased or decreased through the execution of smart contracts or other transactions. In the following, we explain both models with a toy example and explain their advantages and disadvantages. These foundations explain the challenges of building privacy-preserving smart contracts within the Beldex ecosystem.

2.1.1 UTXO Model

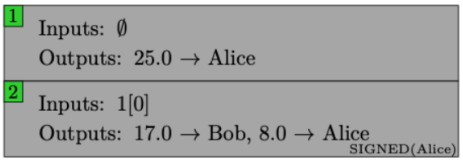

Figure 1: Transactions in the UTXO model

The UTXO model can be illustrated through Figure 1, which shows two transactions. The first transaction generates a new set of coins and has no inputs. Instead, it sends the coins to the address

of Alice, who owns the coins. The second transaction is initiated by Alice, who wants to send some of her coins to Bob. To do this, she references the previous transaction that generated the coins, indicating that she has the authority to spend them. The second transaction sends 17 coins to Bob and returns 8 coins to Alice as change. For the transaction to be valid, it must be signed by Alice using her private key, which ensures that she is the rightful owner of the coins being spent. This process of referencing previous transactions and generating new outputs that can be spent in future transactions is the basis of the UTXO model. One of the central properties of this model is that all inputs must always be spent. The model is called the UTXO model because it is sufficient to keep track of all unspent transactions to model the current state of the system/blockchain. We summarize the properties of the UTXO model as follows:

Input / Output consumption In the UTXO model, each transaction in a blockchain network is composed of one or more inputs and outputs. An input refers to an unspent output of a previous transaction, and an output is a new set of coins or tokens that are being transferred to a new address.

In this model, all inputs must be spent because they represent the unspent outputs of previous transactions. When creating a new transaction, the inputs must reference the unspent outputs of previous transactions that are being used to fund the transaction. Once these inputs are used, they are marked as spent and cannot be used again.

Parallel processing The UTXO model allows for parallel processing of transactions because each transaction in the model is processed independently of all other transactions, making it possible to execute multiple transactions simultaneously.

This parallel processing capability is a key advantage of the UTXO model over other transaction models, such as the account-based model, which requires a global state to be maintained and updated for each account in the network. With the UTXO model, there is no need for a global state, which reduces the risk of bottlenecks and improves scalability.

2.1.2 Integration of Smart Contracts in the UTXO Model.

Integrating smart contracts in the UTXO (Unspent Transaction Output) model can be difficult due to several reasons, including:

Lack of native scripting support The UTXO model was not originally designed to support scripting, which is required for implementing smart contracts. While scripting has been added to the model in later versions, it is still limited in functionality compared to other models like the account-based model. This can make it difficult to implement complex smart contract logic.

Statelessness The UTXO model is a stateless model, which means that there is no concept of an account balance or global state that can be used to store information about the current state of a smart contract. This can make it challenging to implement smart contracts that require stateful operations.

Difficulty in managing contract state Since the UTXO model does not have a global state, managing the state of a smart contract can be difficult. This is because the state of a smart contract needs to be stored in the blockchain, but updating the state requires creating a new output that spends the old one. This can be a complex and inefficient process.

Complex transaction verification Smart contracts in the UTXO model require complex transaction verification logic to ensure that the contract's rules are followed. This can make the verification process more computationally expensive and resource-intensive, which can lead to slower transaction processing times.

2.1.3 The Account-based Model

The account-based model is a transaction-based model used in blockchain systems such as Ethereum, where each account has a balance and is identified by an address. In this model, transactions are executed by transferring funds from one account to another. The operation of the account-based model can be described with the following steps:

- Each user has a unique account, identified by a public address. The address is derived from the user's public key, and anyone can send funds to that address.
- Each account has a balance, which is the sum of all the incoming transactions minus the sum of all the outgoing transactions. The balance is denominated in the cryptocurrency used by the blockchain network.
- To initiate a transaction, the sender creates a transaction object that includes the recipient's address, the amount of cryptocurrency to be transferred, and a transaction fee.
- The transaction is then broadcast to the network and added to a pool of unconfirmed transactions.
- Miners in the network select transactions from the pool and add them to a new block. They verify that the transaction is valid by checking that the sender has sufficient funds in their account and that the transaction fee is adequate.
- Once the block is added to the blockchain, the transaction is considered confirmed, and the funds are transferred from the sender's account to the recipient's account.
- The sender's account balance is reduced by the amount of cryptocurrency transferred plus the transaction fee, while the same amount increases the recipient's account balance.
- Each node on the network maintains a copy of the ledger that records all the account balances and transactions. This ensures that all nodes have the same view of the blockchain and can verify the authenticity of transactions.

Overall, the account-based model provides a simple and efficient way to execute transactions on the blockchain network. It is particularly well-suited for applications that involve frequent transfers of cryptocurrency, such as payments or remittances.

2.1.4 Integration of Smart Contracts in the Account-based Model

Smart contracts are integrated into the account-based model using "account contracts". An account contract is a smart contract associated with an, e.g. Ethereum account. When a transaction is sent to an account with an associated account contract, the smart contract code is executed along with the transaction. In general, the following steps explain the integration and execution of smart contracts:

- A user creates a smart contract and deploys it to the network. The contract code is stored on the blockchain and can be executed by any node on the network.
- The user creates an account contract and associates it with their account. This is done by sending a transaction that deploys the contract to their account address.
- The account contract code specifies the rules and conditions for the account. For example, it might specify that certain transactions can only be executed if they meet certain conditions or if a certain time has elapsed.
- When a transaction is sent to the user's account, the account contract code is executed along with the transaction. This allows the smart contract to enforce the specified rules and conditions.
- The account contract can modify the account's state, which includes the account balance, storage variables, and other account data. The contract can also send transactions to other accounts or call other contracts.
- When a transaction is executed by an account contract, the transaction fee is paid using the account's balance. If the balance is insufficient, the transaction is rejected.
- Each node on the network maintains a copy of the blockchain, including the smart contract code and the associated account contracts. This ensures that all nodes have the same view of the blockchain and can execute transactions consistently.

In summary, the account-based model natively supports smart contracts due to global state property.

2.2 Proof-of-Concept for Beldex: the Hybrid Model

Beldex is a cryptocurrency that was forked from Monero, meaning that it shares many similarities with Monero, including using the underlying Unspent Transaction Output (UTXO) model and ring confidential transactions. In this model, every transaction creates new unspent transaction outputs, which can then be used as inputs in future transactions. This provides a high level of privacy and security, as it makes it difficult for third parties to trace the flow of funds on the network.

Ethereum is a blockchain platform that allows developers to create decentralized applications (dApps) using smart contracts. Beldex aims to incorporate the full functionality of Ethereum into its network. To achieve this goal, Beldex plans to build a hybrid solution that allows users to send BDX tokens to an account-based ledger by burning the coins. This means that the coins will be destroyed in the process, but the user will be credited with an equivalent amount of BDX on the account-based ledger. This will allow BDX to be used for smart contract operations on the account-based ledger. The account-based ledger will be used to perform all smart contract operations on the Beldex network. The hybrid approach will allow Beldex to bring together the best of both worlds: high-efficiency and parallel executions for the financial BDX transaction and the rich functionality of Ethereum-based smart contracts.

Integrating Ethereum code into the native blockchain of Beldex means that Beldex aims to incorporate the full functionality of Ethereum into its network. This is the starting point of the proof of concept, as it allows Beldex to test and demonstrate how it can perform smart contract operations using its own blockchain.

To achieve this, Beldex developers must integrate the Ethereum Virtual Machine (EVM) into the Beldex blockchain. The EVM is the runtime environment that executes smart contracts on the Ethereum network. By integrating the EVM, Beldex could execute the same smart contracts that can run on the Ethereum network.

Once the EVM is integrated, Beldex can begin testing and developing its own smart contracts using Ethereum code. This is the proof of concept phase, where Beldex developers can demonstrate that their blockchain is capable of executing smart contracts in a way that is secure and efficient. The integration of Ethereum code into the native blockchain of Beldex is a significant step towards creating a fully-functional hybrid solution, where BDX can be sent to an account-based ledger to perform all smart contract operations.

2.3 Open Research Problems, Conclusion, and Outlook

The first step towards building this POC is to answer the fundamental question if basic Solidity code, which is the underlying programming language used within Ethereum, can be embedded in a Beldex transaction or not. If it turns out that this would be infeasible, then a hard fork would be needed. Since every hard fork may involve great risks, the first logical step is to determine whether the integration is possible or not. However, even if this is possible, some fundamental questions regarding the execution of the hybrid approach must be addressed by future research:

State transitions UTXO $\rightarrow$ account: One obstacle of this hybrid approach is state transition between both ledgers. The transition from the UTXO-based ledger to the account-based ledger seems to be realizable by burning coins. I.e., whenever a user wishes to add coins to its account, then it presents a proof of burn. In this regard, future research will develop solutions that guarantee that each proof of burn can only be used once.

State transitions account $\rightarrow$ UTXO Another open research problem is the conversion between coins stored on the account-based ledger and transferred back to the UTXO ledger. The challenging part here seems to be that the transaction must be valid w.r.t. to the ring CT protocol. Embedding of a private key within the smart contract raises the issue that anybody could steal this key, create a ring CT transaction, and steal the funds. One approach to solve this issue might be the integration of the masternodes that are used in Beldex. The advantage of including the masternodes is that each masternode staked 10,000 BDX and therefore reaches a higher trust level than any "regular" user.

Future research will address these technical challenges.

3 Security and Privacy Goals

Beldex is a comprehensive platform that provides a complete privacy-based ecosystem for its users. The platform achieves this by employing various cutting-edge technologies such as anonymization, incentivization, and an economic model to ensure the sustainable development of the ecosystem. The main objective of the platform is to anonymize transactions, messages, and the online footprint of users.

The private ecosystem comprises various components, including the Beldex blockchain, decentralized and anonymous messenger, browser, wallet, a privacy protocol, and the Beldex Bridge,

which connects the platform to other ecosystems. These components work together seamlessly to provide users with a completely private online experience.

One of the areas that Beldex is actively researching is the extension of its ecosystem to support smart contracts. The platform is committed to ensuring that smart contracts are executed in a privacy-preserving manner, which aligns with the platform's core vision of protecting users' privacy at all times. Beldex believes that the privacy of its users should not be compromised, even when executing smart contracts.

The concept of privacy-preserving smart contracts is a promising idea, but there is still a lack of clarity surrounding what it actually means. To better understand and realize the potential of privacy-preserving smart contracts, it is important to recognize some of the main challenges that exist in this area.

One of the main challenges is the complexity of smart contracts themselves. Smart contracts are self-executing code that is embedded within a blockchain network. They are designed to automate complex processes and transactions, but the intricacies of these contracts can make it difficult to ensure that privacy is being maintained throughout the process.

Another challenge is the need for privacy while still ensuring that the smart contract is able to function properly. In some cases, privacy measures may hinder the ability of the smart contract to execute as intended, which can lead to errors and potential vulnerabilities. This means that privacy-preserving smart contracts must strike a delicate balance between ensuring privacy and maintaining functionality.

Another challenge is balancing the benefits of privacy with the potential risks of malicious actors exploiting the privacy measures. While privacy is important, it is also important to ensure that the system is able to detect and prevent fraudulent or malicious activity.

In conclusion, privacy-preserving smart contracts represent an exciting opportunity for enhancing privacy and security within blockchain networks. In the following, we discuss some of the open research directions and possible interpretations of "privacy-preserving smart contracts" in this area.

Hiding the Program One possible interpretation of privacy-preserving smart contracts is that the program itself is hidden. In other words, the contract code is not publicly visible but stored encrypted on the chain. If one is willing to follow this idea, it raises the question of how the nodes can execute this program. One approach might be that the input to the program contains the decryption key, the code is then decrypted and executed, and the user obtains the result of the local computation. Such an approach seems challenging, as the question of, e.g., how the decryption key is shared is unclear.

Hiding the Inputs An alternative approach of privacy-preserving smart contracts is one that leaks the program, but the inputs and outputs remain hidden. Similar approaches are known in the area of secure two- and multi-party computation. One example of such as Yao's garbled circuit protocol [Yao82]. Garbled circuits are a technique that can be used to maintain privacy in computations by obscuring the input values while still allowing the computation to be performed accurately. This is achieved by encoding the inputs using encryption and then passing them through a "garbled circuit," which assigns random values to the input and output wires of the circuit and encrypts those values so that they cannot be decoded without the appropriate keys. The encrypted values are then sent to the other parties involved in the computation, who use their keys to decode the values and perform the necessary computations on the inputs. Garbled circuits are a powerful tool for maintaining privacy in

various applications, including secure data sharing and privacy-preserving analytics.

Adapting this idea to the setting of smart contracts seems challenging as well because garbled circuits can only be executed once, and also, the distributions of the roles within the protocol are unclear.

3.1 Known Approaches For Privacy-Preserving Smart Contracts

3.1.1 The Hawk Protocol

In the following, we summarize the main idea of Hawk protocol by Kosba et al. [Kos+16b]. The basic idea of their approach is a compiler that compiles a given program into a cryptographic program that is run between the blockchain and the user (with the help of some minimally trusted manager) as shown in Figure 2.

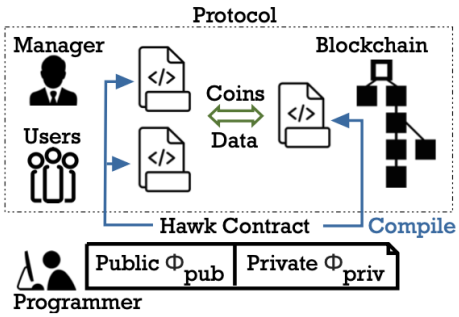

Figure 2: Overview of Hawk [Kos+16b].

The compiled program distinguishes between private and public data. The private portion may contain sensitive information, such as e.g. a bet. Public information should not touch any financial aspects. The compiled program consists of three sub-programs, one for the user, one for the parties running the consensus protocol, and one for the manager. The resulting protocol achieves the following privacy guarantees:

On-chain Privacy The concept of on-chain privacy ensures that transactional privacy is maintained from the public, unless the parties involved in the contract willingly disclose information. In Hawk protocols, users interact with the blockchain to ensure fairness and prevent aborts, but the monetary flow and transaction amounts in the private Hawk program are cryptographically concealed from public view. This is achieved through the use of "encrypted" information sent to the blockchain, and enforced by zero-knowledge proofs to ensure accurate execution of the contract and proper conservation of funds.

Contractual Privacy Contractual security ensures protection between parties involved in an agreement. In Hawk, parties are expected to prioritize their financial interests, leading to deviations from the protocol or early termination. To address this, contractual security includes confidentiality, authenticity, and financial fairness to prevent cheating and premature

termination. To gain a better understanding, readers can refer to Section I-B, which provides a specific example.

3.1.2 The Eagle Protocol

The Hawk protocol [Kos+16b] introduced the concept of general-purpose privacy-preserving smart contracts. This required users to confidentially submit input strings and balances to a trusted contract manager. After the contract is evaluated over these private inputs, the contract manager settles confidential outputs to a confidential ledger while providing zero-knowledge proof that the outputs have been obtained according to the contract's instructions. The authors of the eagle protocol [Bau+22] recognized three challenges:

- Efficiently distributing the role of the trusted third party to avoid a single point of failure without compromising performance.
- Requiring clients to be online only during a brief input phase, similar to the standard client-blockchain interaction model where clients broadcast signed inputs.
- Enabling privacy-preserving smart contracts to be long-running applications over indefinite rounds, as seen in standard public smart contracts.

The authors present "Eagle", a protocol that achieves efficient privacy-preserving smart contracts while addressing the challenges of a distributed trusted third party, offline clients, and long-running applications. Their approach uses outsourced secure multi-party computation and insured MPC to incentivize fairness and security. They also utilize a novel input protocol and a reactive version of their MPC protocol to maintain a secret off-chain state over multiple rounds, allowing for continuous, multi-round evaluation even when clients are offline.

3.2 The Security and Privacy Goals of this PoC

Beldex is aware of the different privacy notions, challenges, and open problems in the context of privacy-preserving smart contracts. Therefore, the first proof of concept only concerns itself with the feasibility of embedding smart contracts within the Beldex Blockchain. This initial stage aims to demonstrate the basic functionality of the system and provide a solid foundation for further development. As the project progresses, further research will explore different notions of privacy and verifiable computation to ensure that the system can support a variety of use cases while maintaining the highest levels of security and privacy. By taking a comprehensive approach to privacy-preserving smart contracts, Beldex can create a system that meets the needs of a wide range of industries and users, while also contributing to the ongoing research and development of privacy-preserving technologies.

3.3 Summary and Outlook

Privacy-Preserving Smart Contracts are an active field of research with ongoing efforts to improve privacy and security in transactions. One of the challenges is to understand what privacy means in the context of smart contracts and how to achieve it effectively. As Beldex builds an ecosystem for privacy-preserving smart contracts, further research will explore this

setting to enable smart contracts that can execute in a privacy-preserving manner. The goal is to support privacy-preserving smart contracts in the Beldex ecosystem by leveraging cutting-edge techniques such as homomorphic encryption, zero-knowledge proofs, and garbled circuits. With these privacy-preserving mechanisms, the Beldex ecosystem can enable secure and private transactions in a variety of industries, including finance, healthcare, and more.

4 System Overview

This section describes the technical realization of smart contract development using the Ethereum Virtual Machine (EVM) for Beldex, a privacy-focused blockchain platform. It begins with an introduction to the basic layout of a Beldex block, followed by an overview of the EVM and its features. The section then discusses related work in the field of smart contract implementation and the design process of the EVM proof of concept (POC) for Beldex. It explains how to create and deploy smart contracts onto the Beldex blockchain and how to use the Contract State Viewer to inspect the state of a deployed smart contract. The section also covers how to trigger a transaction using an EVM event and the different ways in which a smart contract can be terminated. Overall, this section provides a technical understanding of the EVM and its usage for smart contract development on the Beldex blockchain.

4.1 The Basic Layout of a Block

Figure 3 displays the high-level structure of a block in the Beldex blockchain network. The block contains regular content, such as a list of validated transactions that have been added to the blockchain. Additionally, the block includes all the necessary information for the Proof of Stake protocol, which ensures that the network's nodes validate the block before it can be added to the chain.

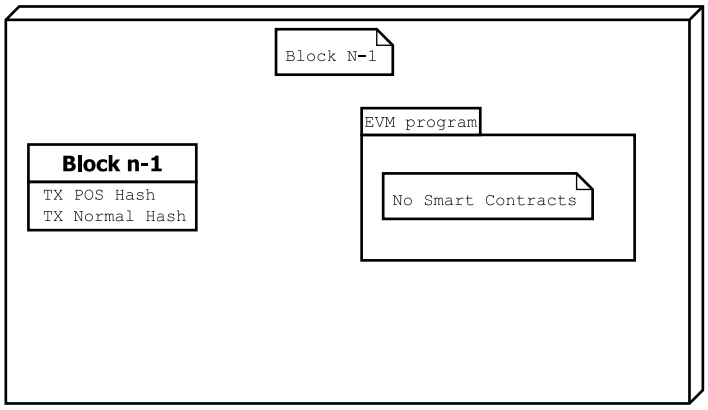

Figure 3: Basic layout of a block

The block will be extended with the EVM program (see Section 4.2), which is a standard program for executing smart contracts on the Ethereum blockchain. By adding the EVM program to the Beldex blockchain, the network will be able to support smart contract functionality, allowing for more complex and decentralized applications to be built on top of the blockchain.

It should be noted that in this example, the n-1 block does not carry any code, meaning that the next block will be the first to include the EVM program and enable smart contract functionality on the Beldex blockchain. Overall, the picture provides an overview of the block's structure and content in the Beldex blockchain network and highlights the addition of the EVM program for

smart contract functionality.

4.2 The Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is an open and distributed virtual computer that is capable of executing Turing-complete programs, also known as smart contracts [Hir17]. These smart contracts can be created in the Solidity language, which is a curly-bracket language designed specifically to target the EVM [Bha+16].

Smart contracts are extremely versatile and can be used to develop a wide range of applications, including voting systems, crowdfunding platforms, blind auctions, and multi-signature wallets.

Although there are distributed applications available on Cryptonote, such as a privacy chat service built on the Beldex framework [Bel], these services are typically created and maintained by the founders of the framework. To enable users to create and manage their own distributed applications on the Beldex framework, an implementation of the EVM is necessary.

Currently, there is no EVM implementation available on any Cryptonote-powered blockchain to the best of our knowledge.

We constructed use cases that reflect the minimal functionality of EVM integration on the Cryptonote layer. We succeeded in the implementation and testing of the described use cases that fit a bare EVM in the Beldex blockchain.

Our main contribution is delivering a proof of concept of a minimized Ethereum Virtual Machine implemented in a Cryptonote Proof of Stake blockchain, specifically Beldex. Which enables other Cryptonote privacy-oriented cryptocurrencies like Monero[Mon14] and Beldex to integrate an EVM in it's own blockchain. The tested sourcecode for our proof of concept [DN22] is available on Github.

4.3 Related Works and the design process of the EVM POC for Beldex

The Beldex blockchain [Bel22] utilizes the Cryptonote [VS13] protocol for its application layer and the RingCT [Sun+17] protocol for transactions and signatures. The original Ethereum Virtual Machine (EVM) [Hir17] serves as the starting point, defining the state and smart contracts, among other variables. One such variable is Gas, which is the dynamic fee for using a smart contract calculated based on the cost of EVM instructions.

Given Cryptonote's focus on financial privacy and traceability, it is essential to maintain privacy when implementing smart contracts on top of it. This means considering the information that can be extracted from contracts, such as the amount of coins or source code. Zero-Knowledge (ZK) protocols can be used to make this information private, as seen in related work like Zether [Bün+20], which implements privacy on the blockchain as a smart contract on top of the EVM using ZK-Snarks [Wah+18] with modifications.

For this study, we aim to develop a proof of concept for an EVM on the Beldex blockchain, using Python-based Py-EVM [Eth18]. The POC will include basic features such as creating and viewing contract states, a "Hello World" contract, and termination of contracts, all on a dedicated EVM with 13 Masternodes responsible for block creation using Proof of Stake. This POC will provide insights towards implementing a full-featured Py-EVM for the Beldex blockchain. While ZK protocols are crucial for maintaining privacy on Cryptonote, this study will focus on implementing the basic mechanism of an EVM on the protocol. Future work can explore ZK smart contracts on Cryptonote.

4.4 Creation of a new contract

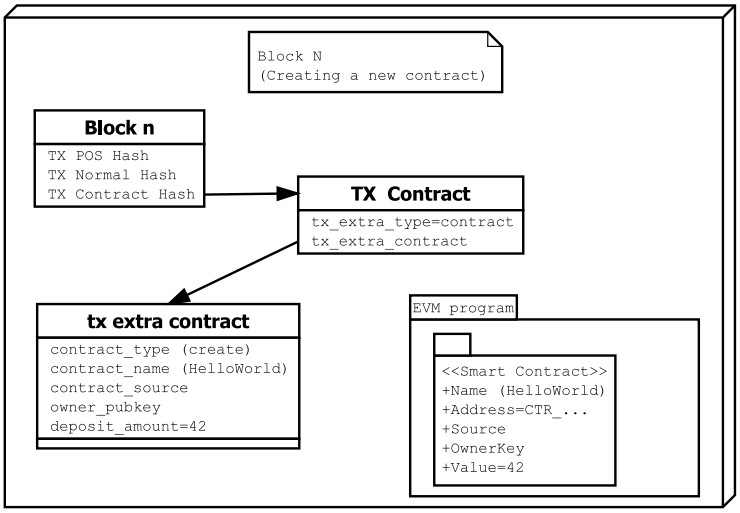

Figure 4: Creating of a smart contract

To begin, a contract creation transaction is a type of transaction that is used to deploy a smart contract on the blockchain network. We describe the basic layout with the help of Figure 4. When a contract transaction is added to a block, it becomes a part of the blockchain and is validated by the network's nodes through the Proof of Stake protocol. This ensures that the contract transaction is legitimate and cannot be tampered with.

To bind the contract to the block, the contract transaction is added to the block along with other transactions that have been validated and added to the blockchain. This is done by including the contract transaction in the list of validated transactions that are stored in the block. Additionally, the contract transaction carries a tx_extra_type of "contract," which indicates that it is a contract transaction and should be treated as such by the network's nodes.

In the running example, the contract transaction carries additional details about the contract, including its type and name, such as "HelloWorld." It also stores the address of the contract's owner and a deposit amount. These details are important because they provide information about the contract that the network's nodes can use to validate the contract and ensure its legitimacy.

Once the contract transaction has been added to the block, it becomes a part of the blockchain network and can be executed by the network's nodes. Note that the contract is not executed once, but all contracts will be executed on any update of a new block. This allows for more complex and decentralized applications to be built on top of the blockchain, providing new opportunities for innovation and development in the blockchain space. Overall, the addition of a contract transaction to a block in the Beldex blockchain network is an important step in enabling smart contract functionality and enhancing the capabilities of the blockchain. The user initiates the command line interface wallet and connects it to the EVM to create a new account with a specified amount of coins. The user can also specify the amount of coins to be sent to the new smart contract during its creation.

To create the new smart contract, the user needs to specify its name, source code, and the

amount of coins to be sent from their current wallet to the new contract. Once the user confirms the contract name, coins amount, contract source code, and a fee, the wallet creates a new transaction that is distributed to all nodes.

Upon successfully accepting the transaction in a new block, the EVM initializes the new smart contract with the sent coins amount and other user-specified values. Note that these sent coins are the initial coins that the contract creator is funding in the new contract. The user is then displayed the address of the newly created contract and the corresponding smart contract keys.

4.5 Contract state viewer

The command line interface provides the option to view the current state of a smart contract by entering its address. The state of a smart contract is updated after every received block, and various values can be extracted from it, including the protocol version, the available coins, the name, the source code, as well as the incoming and outgoing transactions. Additionally, the internal state of the contract, such as: idle, running, or waiting for transactions, can also be viewed.

4.6 A Simple "Hello World" Contract

We believe that it is not necessary to provide actual EVM byte code compiled with Solidity to demonstrate the concept of a working EVM and its integration in the Beldex framework. The source code for this simple contract is literally just "Hello World". The parsing of EVM instructions is not necessary at this stage. However, any optional parameters provided will be printed in the terminal when a user calls a method of this contract. The Python EVM console will print out "Hello World" when the contract's method is triggered.

4.7 Transaction Triggered by an EVM Event

Figure 5 shows the individual steps of a process involving the structure of the block content, the execution of the EVM program, and the addition of a new transaction to the TX pool. In the upper part of the figure, the structure of the block content is shown, which includes the block header, transaction counter, transaction list, and the state tree root. Below this, the execution of the EVM program is depicted, with the program calling the method "split" with a value of 12. The result of this method execution is the creation of a new state tree, which is reflected in the updated state tree root in the block content. Finally, at the bottom of the figure, a new transaction of value 6 is added to the TX pool, which will be included in a future block. Overall, these steps demonstrate the process of executing a smart contract method and updating the state of the blockchain, as well as the addition of new transactions to the TX pool for future inclusion in the blockchain.

The ability to trigger a transaction based on a dynamic condition set in the EVM contract is the most important and complex capability in this proof of concept. For example, we can create a smart contract that automatically transfers 50% of the received amount to a fixed address and keeps the other 50% in the contract's coin value.

4.8 Termination of a Contract

During development, it is desirable to have the ability to terminate a contract with a clean-up that removes as much as possible. However, since the blockchain keeps track of history, it is not possible to fully remove a contract. Therefore, the contract should be disabled upon termination, and any

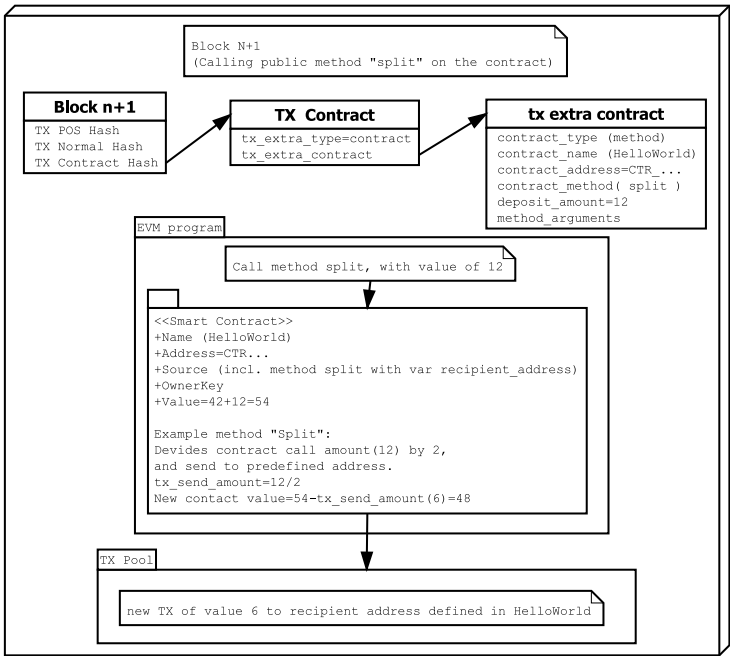

Figure 5: Calling the public method “split”.

potential available coins should be returned to the owner. It is also important to consider what should be done when a user sends coins to a terminated contract. We suggest that these coins be sent back to the sender, minus a fee for the transaction.

4.9 Smart Contract values stored in TX Extra

This section describes the structure of the TX extra fields for a smart contract and we refer the reader to Figure 4 for a high-level description. The TX extra field is stored in the block chain and looks like this in the current iteration of the POC:

```
struct tx_extra_contract
  {
      contract::contract_type m_type;
      uint8_t m_version=4;
      std::string m_contract_name;
      account_public_address m_contract_address;
      crypto::public_key m_owner_pubkey;
      crypto::secret_key m_spendkey;
      crypto::secret_key m_viewkey;
      std::string m_contract_source;
      std::string m_contract_method;
      std::string m_method_args;
      uint64_t m_deposit_amount;
      crypto::signature m_signature;
```

}

In the first iteration of the EVM POC development, we experimented with burning the coins amount that was initially sent to the contract during the creation of the contract. However, the burning of the coins itself was a success we couldn't transfer these coins to another address because of the complexity of the validity checks. To further develop the EVM POC and to discover possible further design problems we experimented with adding private keys and succeeded in transferring coins from the contract. While this gave insight into further development, we also introduced a new problem that the private keys can easily be taken from the block chain and used for spending by others. To solve this problem, we added a validation rule that checks that a transaction really originates from a smart contract when coins are sent from a smart contract. While this works for the current proof of concept, we need an approach without using private keys in the chain. Private keys are private and should never be available in public. Another discovered problem using this approach is using a wallet for each contract that will create significant load on the nodes reducing the scalability of the network.

5 Evaluation of the EVM

By analyzing the prior created post conditions of the use cases we evaluated the expected results. Testing the EVM is done on 16 synced proof of stake nodes on a dedicated EVM, the command line interface for the wallet connected to a local node, a web-GUI for exploring the state of the blocks and the EVM-Python CLI to show the state of the EVM. We manually tested the use cases during each development iteration. This test process was done by using the wallet command line interface to create a smart contract and by checking on a different remote node to accept a new block that includes the created smart contract. Immediately after accepting the block with a new smart contract or a method call of this contract, the EVM is triggered and is checked for acceptance of the use case. The evaluation gave insight in newly discovered problems like the double spending complexity and privacy for smart contracts.

We created a stub protection in the node to handle the double spending problem that is triggered when coins are trying to be spend that belong to a smart contract. The stub currently only shows a message that the current transaction is originating from a smart contract and that the EVM must be checked if this transaction is really originated from the EVM. While this mechanism can be implemented like this, it gave insight it is desired to solve this problem in the node itself. One possible solution to try in the next iteration is to store allowed transactions in the memory pool of the node when the EVM is ready so that the node does not need to check the permission in the EVM and hereby save processing resources. The other discovery issue is on the privacy of smart contracts and probably does not need to be seen as a problem. We discuss both the issue of double spending and privacy in the next sections.

5.1 Design of the transaction triggered by an EVM event and the double spending problem

Key images contain encrypted information with coins transferred from one address to another, only readable by the keyholders. The privacy of these keyimages is enhanced with the implementation of a keyring. The keyring consists of 10 signatures, of which 9 are fake, to protect the trace of the transaction. A simple user-triggered transfer from the wallet needs the private and public

view and spend keys. The keys are needed to scan the chain for available coins to the user in the complete set of all transactions and to create the valid signatures in the keyimage that is send to the receiver. Simple stated, the implemented keyimage with the keyring contain coins and is linked to an address, that are only viewable for the involved parties.

To make it possible for the EVM to transfer coins it needs to be able to create a key image that is spendable by the receiver. We first experimented with an approach to transfer coins from a user to the EVM contract by a method called burning. When coins are burned, the input key image is set as spent and the transfer output does not exist. We triggered this by detecting an EVM address in the address's prefix. In this case the selected coins are added to the state of the EVM contract and not added to a new key image. This way all nodes hold the coin value in the EVM state, intending that the EVM must know the value for it's processing capability. We succeeded in burning the value for the sender and listing the coin value in the EVM state using this method. However we encountered a problem with spending the coins from the EVM contract. When trying to send coins from the EVM value to another user address the nodes check if the transfer is valid by checking if correct keyimages are used as input by the spender.

To make this possible the verification for spending coins by a EVM needs to be added. Because no input keyimage is available another way of storing value needs to be available. In the POC the value of a contract can be retrieved from its dynamic state. It will help if the value of contracts are available in the same database as the keyimages, the LMDB[How11]. When this value is available in the database, the node does not need to request the value from the separate EVM as it is implemented now and will be better for performance. However, we skipped this in our implementation because we believe this must be implemented when the EVM becomes in production. To make it possible for the EVM to send coins from a contract to another address, it needs to know the keys of the available coins. A contract is identified by a contract address that concatenates a contract base58 prefix "Ctr", the public view key, and the public spend key. During the contract creation, this contract address with corresponding private keys for this contract is generated using the same method as a new account is generated for a wallet.

6 Privacy of the contract or the contract owner?

While developing our solution for double spending, we primarily focused on ensuring its effectiveness. However, we realized that we had not given adequate attention to the privacy of the smart contracts. In our next iteration of development, we plan to prioritize the financial privacy of these contracts.

While a normal transfer of coins that is sent to another address is typically private, the privacy of a smart contract depends on several factors. One of these factors is the viewability of the smart contract source code, as well as the available coins for everyone.

One of the challenges in designing a smart contract that prioritizes privacy is that the contract needs permission to execute transactions by itself without the need for interaction from a smart contract owner. This is particularly important for contracts that involve sensitive financial information.

To address this design problem, we propose the creation of a smart contract that has two distinct parts: one that needs to be visible to all users and another that is private and spendable to protect the amount of coins. By separating the contract into these two parts, we can ensure that sensitive financial information remains private while allowing the necessary permissions to execute

transactions autonomously.

Overall, prioritizing financial privacy in smart contract development is crucial for ensuring the security and protection of sensitive financial information.

Is it possible to prove an account holds certain funds without revealing its private values? With normal transactions, this is managed by the use of RingCT proofs. The sum of input+output values is non-zero in RingCT to cryptographically protect, and this proves funds are not created from thin air. For sending funds, the amount is added to the cipher using the public key of the receiver. Private keys are needed to split the inputs into multiple outputs, making one for the correct sending amount and the rest the change to another output. This problem needs to be solved in future work.

7 Conclusions

The approach of designing a proof of concept for an Ethereum Virtual Machine gave insight in the complexities of a double spending solution and issues with privacy for smart contracts. The development of the Proof of concept made clear that all EVM's in the network need to be deterministic in the way how a transaction is stored and distributed over the network to make it possible for a good validation.

While the byte code of contracts needs to be readable by the Virtual Machine, the side effect is that the byte code is viewable for everyone. On the other side, the source code of the nodes is also always readable. And the privacy of the contract holder or the receiver or originator of the coins is kept private.

While Zero Knowledge contracts seem to be possible, this needs further research before integration on top of the Cryptonote application protocol that is used in Beldex.

Acknowledgment

The author would like to personally thank Martijn Nuijens for his efforts and support in developing the EVM Proof of Concept.

References

[And+13] Elli Androulaki et al. "Evaluating User Privacy in Bitcoin". In: *FC'13*. 2013.

[Bar+12] Simon Barber et al. "Bitter to Better. How to Make Bitcoin a Better Currency". In: *FC'12*. 2012.

[Bau+22] Carsten Baum et al. *Eagle: Efficient Privacy Preserving Smart Contracts*. Cryptology ePrint Archive, Paper 2022/1435. `https://eprint.iacr.org/2022/1435`. 2022. URL: `https://eprint.iacr.org/2022/1435`.

[Bel] *Beldex - Private and Decentralized Ecosystem*. 2022. URL: `https://www.beldex.io/whitepaper.pdf`.

[Bel22] Beldex. *Beldex Sourcecode*. 2022. URL: `https://github.com/Beldex-Coin`.

[Bha+16] Karthikeyan Bhargavan et al. "Formal verification of smart contracts: Short paper". In: *Proceedings of the 2016 ACM workshop on programming languages and analysis for security*. 2016, pp. 91–96.

[Bon+14] Joseph Bonneau et al. "Mixcoin: Anonymity for Bitcoin with Accountable Mixes". In: *FC'14*. 2014.

[BS+14] Eli Ben-Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *S&P'14*. 2014.

[Bün+20] Benedikt Bünz et al. "Zether: Towards Privacy in a Smart Contract World". In: *Financial Cryptography and Data Security*. Ed. by Joseph Bonneau and Nadia Heninger. Cham: Springer International Publishing, 2020, pp. 423–443. ISBN: 978-3-030-51280-4.

[DN22] Allard Dijk and Martijn Nuijens. *EVM POC Sourcecode*. 2022. URL: `https://github.com/Beldex-Coin/evm-poc`.

[Eth18] Ethereum. *Python Implementation of the EVM*. 2018. URL: `https://github.com/ethereum/py-evm`.

[GK15] Jens Groth and Markulf Kohlweiss. "One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin". In: *EUROCRYPT'15*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 253–280. DOI: `10.1007/978-3-662-46803-6_9`.

[Hei+17] Ethan Heilman et al. "TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub". In: *NDSS'17*. 2017.

[Hir17] Yoichi Hirai. "Defining the Ethereum Virtual Machine for Interactive Theorem Provers". In: *Financial Cryptography and Data Security*. Ed. by Michael Brenner et al. Cham: Springer International Publishing, 2017, pp. 520–535. ISBN: 978-3-319-70278-0.

[How11] Chu Howard. *Lightning Memory-Mapped Database Manager*. 2011. URL: `http://www.lmdb.tech/doc/`.

[KKM14] Philip Koshy, Diana Koshy, and Patrick McDaniel. "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic". In: *FC'14*. 2014.

[Kos+16a] Ahmed E. Kosba et al. "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: *S&P'16*. 2016.

[Kos+16b] Ahmed E. Kosba et al. "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 839–858. DOI: `10.1109/SP.2016.55`.

[Max13] Greg Maxwell. *CoinJoin: Bitcoin Privacy for the Real World*. Post on Bitcoin Forum. `https://bitcointalk.org/index.php?topic=279249`. 2013.

[Max15] Greg Maxwell. *Confidential Transactions*. `https://people.xiph.org/~greg/confidential_values.txt`. 2015.

[Mei+13] Sarah Meiklejohn et al. "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names". In: *IMC'13*. 2013.

[Mie+13] Ian Miers et al. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". In: *S&P'13*. 2013.

[MO15] Sarah Meiklejohn and Claudio Orlandi. "Privacy-Enhancing Overlays in Bitcoin". In: *BITCOIN'15*. 2015.

[Mon14] Monero. 2014. URL: https://getmonero.org/.

[Mon23] Cryptocurrency market capitalizations Monero. visited Feb, 5th 2023. 2023. URL: https://coinmarketcap.com/currencies/monero/.

[Poe+17] Andrew Poelstra et al. "Confidential Assets". In: *BITCOIN'17*. 2017.

[RH13] Fergal Reid and Martin Harrigan. "An Analysis of Anonymity in the Bitcoin System". In: *SXSW'13*. 2013.

[RMS17] Tim Ruffing and Pedro Moreno-Sanchez. "ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin". In: *BITCOIN'17*. 2017.

[RMSK17] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "P2P Mixing and Unlinkable Bitcoin Transactions". In: *NDSS'17*. 2017.

[Sab13] Nicolas van Saberhagen. "CryptoNote v 2.0". In: (2013). https://cryptonote.org/whitepaper.pdf.

[SMZ14] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. "BitIodine: Extracting Intelligence from the Bitcoin Network". In: *FC'14*. 2014.

[Sun+17] Shi-Feng Sun et al. "RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero". In: *Computer Security – ESORICS 2017*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Cham: Springer International Publishing, 2017, pp. 456–474. ISBN: 978-3-319-66399-9.

[VR15] Luke Valenta and Brendan Rowan. "Blindcoin: Blinded, Accountable Mixes for Bitcoin". In: *BITCOIN'15*. 2015.

[VS13] Nicolas Van Saberhagen. "CryptoNote v 2.0". In: (2013).

[Wah+18] Riad S. Wahby et al. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)". In: *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164.

[Zca16] Zcash. 2016. URL: https://z.cash/.

[Zco16] Zcoin. 2016. URL: https://zcoin.io/.

[Zie+15] Jan Henrik Ziegeldorf et al. "CoinParty: Secure Multi-Party Mixing of Bitcoins". In: *CODASPY'15*. 2015.