A trust-minimized e-cash for cryptocurrencies

Mario Yaksetig

BitFashioned SEZC
mario@bitfashioned.com

Abstract. We introduce a private cryptocurrency design based on the original e-cash protocol. Our proposal allows for private payments on existing blockchain systems. In our design, the issuance of the private cash is transparent and is associated with a blockchain transfer to provide stronger security.

Keywords: E-Cash · Privacy · Blockchain.

1 Introduction

In the realm of digital finance, the concept of e-cash emerged as a pivotal innovation, offering a digital counterpart to traditional physical cash. This technology promises expedited and convenient transactions, while providing privacy in the payments that take place. However, traditional e-cash systems often hinge on very opaque central authorities for asset issuance and transaction validation. A trust-minimized e-cash system seeks to add transparency to this setting.

1.1 Background

Created by Chaum [5], e-cash represents the first instantiation of digital cash. This solution, which consists of a withdrawal and payment protocol, allows for private payments via the use of blind signatures upon a withdrawal process. Essentially, users have funds which they convert (or withdraw) into a private representation of those assets. This private version of the funds is comprised of multiple denominations which can be privately spent. Improvements to the original e-cash have been proposed in [3, 4, 2, 6] and decentralized versions proposed in [11, 1, 10].

1.2 Motivation

The goal of this work is to allow for a modular 'add-on' system for private payments on top of existing cryptocurrencies. Using well-established primitives such as blind signatures [5] and trying to mitigate the inherent centralization problems with the original e-cash, we aimed at building a trust-minimized e-cash system that operates on top of an underlying blockchain. This is particularly motivated by the rise of rollups, which are particularly well-suited for this design.

1.3 Our contributions

We introduce a trust-minimized variant of the original e-cash proposal. In our approach, it is possible for anyone to confirm whether or not a specific issuance corresponds to a legitimate withdrawal, which completely mitigates the ability for a rogue issuer to create money out of thin air. This feature is possible to achieve due to the fact that the blockchain contains a public record of all the withdraw transactions that take place. These transactions reveal the identity (i.e., the wallet address) of the person performing the withdrawal, but does not leak the coin identifier, thus preserving the privacy properties of the original e-cash.

2 Our Construction

In this section, we cover our proposed construction. We start with a simple overview of the protocol. We then cover the architecture of the system and the detailed protocol description.

2.1 Protocol Overview

Our construction is very similar to the original proposal from Chaum [6]. We, however, assume that users communicate with a smart contract to perform withdrawals as opposed to communicating with a commercial bank (in the indirect model) or a central bank (in the direct model). Our design is simple: to convert (withdraw) cryptocurrency funds into private e-cash, user must perform a debit transaction from their wallets and publish the blinded coin along with the corresponding transaction. This ensures that it is possible to monitor exactly how much money is in circulation and ensure that the minting of money is verifiable. In any e-cash based systems, the (blind) signing process is traditionally opaque. This lack of transparency is not aligned with the core principles and ethos of decentralized systems.

2.2 System Architecture

Our system comprises the following entities: users, merchants, a blockchain with smart contract capabilities, and an asset issuer.

User(s). We consider the existence of users in the system that want to perform payments using the crypto e-cash. Users start their flow by performing a withdrawal and converting their funds to the private version of such funds. Upon successful withdrawal, users are able to make private payments. Therefore, users perform both the withdrawal and payment protocols (see Figure 2 and 3 for more detail).

Merchant(s). We consider the role of a merchant that is in charge of processing payments directly from users. When receiving a payment from users, the merchants check with the issuer if the coin is properly signed and are able to redeem that amount directly into their wallets.

Smart Contract. We consider a smart contract running on a layer-1 blockchain. This contract is secured by the consensus of such blockchain and processes transactions that come from either users or merchants. Transactions from users imply a debit and a corresponding blinded coin. Transactions from merchants imply a credit transaction that adds balance to a specific wallet.

Issuer. We assume the existence of an off-chain issuer. By off-chain, we mean an issuer that is an external entity that does not run at a smart contract or layer-1 (L1) level. The issuer is in charge of scanning through the new coin issuance requests that are posted on the blockchain and sign them with the corresponding issuance keys for each denomination.

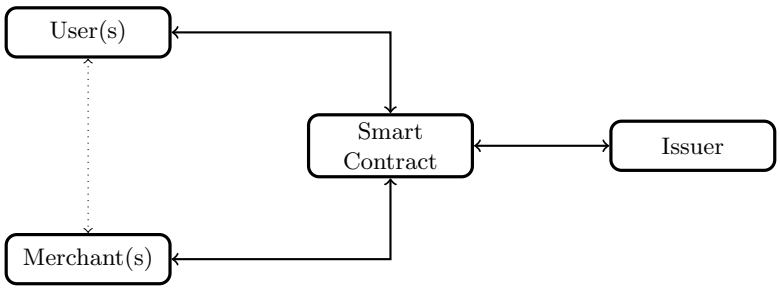

Fig. 1. Caption

2.3 Adversarial Model

We assume an adversary $\mathcal{A}$ that is focused on disrupting the system and breaking the privacy of withdrawals and/or payments. The goal of the adversary is to disrupt the system by either compromising the privacy of the issued coins, successfully executing transactions that double spend funds, and/or issuing counterfeit coins without detection. To do so, $\mathcal{A}$ may compromise different entities in the system. The adversary is assumed to control some users and merchants. However, we assume a minimum number of honest users and merchants. We do not consider adversaries whose purpose is to launch denial-of-service (DoS) attacks.

2.4 Protocol Description

In this section, we describe our design. We note that our proposal consists of two protocols: a withdrawal protocol and a payment protocol.

Withdrawal Protocol. This protocol consists of the following algorithms:

- $\mathsf{Keygen}(1^\lambda) \rightarrow (\mathsf{sk}, \mathsf{pk})$. Generates a secret key value sk given a security parameter λ and its corresponding public-key $\mathsf{pk} = g^{\mathsf{sk}} \mod p$. Returns the key pair.
- $\mathsf{CoinGeneration}(1^\lambda) \rightarrow (x, \mathsf{c})$. Generates a secret preimage value x given a security parameter λ. Upon generation of the preimage, Alice obtains the coin c by hashing the preimage $\mathsf{c} \leftarrow \mathcal{H}(x)$. This hash function maps random values to group elements. Therefore, $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{F}_p$.
- $\mathsf{BlindCoin}(\mathsf{c}) \rightarrow \mathsf{bc}$. Given a coin c, this algorithm creates a random blinding factor r and returns a blinded coin $\mathsf{bc} = \mathsf{c}^{\mathsf{r}} \mod p$
- $\mathsf{UnblindCoin}(\mathsf{bc}, \mathsf{r}) \rightarrow \mathsf{c}$. Given a blinded coin bc and blinding factor r, this algorithm calculates r^{-1} and returns a blinded coin $\mathsf{c} = (\mathsf{c}^r)^{\mathsf{r}^{-1}} = \mathsf{c} \mod p$.
- $\mathsf{BlindSign}(\mathsf{sk}, \mathsf{bc}) \rightarrow (\sigma^*, \pi)$. Given a blinded coin bc and a secret key sk, this algorithm returns a signed (blinded) coin $\sigma^* = (\mathsf{bc})^{\mathsf{sk}} \mod p$ along with a zero-knowledge proof π of correct exponentiation.
- $\mathsf{VerifySignature}(\mathsf{pk}, \sigma^*, \pi) \rightarrow \mathsf{Ok}/\mathsf{NotOk}$. Given a public-key pk, a (blind) signature, and a proof of correct exponentiation, this algorithm returns Ok if σ^* is properly 'signed' with sk or NotOk if not properly signed.

To withdraw funds, user Alice must first have v funds in her account (i.e., wallet). First, the user randomly generates a preimage and corresponding coin identifier (hash). Second, Alice obtains a blinded version of said coin. Third, Alice uploads the blinded coin to the blockchain along with a debit transaction. The issuer, upon seeing that a new debit took place on the blockchain, signs the blinded coin and posts the proof of correctness. This mapping ensures that new money is only issued when a debit transaction takes place on the blockchain. Alice, checks that the proof is correct and unblinds the coin, which is now ready to be spent. This finalizes the protocol.

We refer the reader to Figure 2 for further detail.

Payment Protocol. This protocol consists of the following algorithms:

- $\mathsf{Pay}(\mathsf{amount}, \mathsf{m_{id}}) \rightarrow (x, \sigma')$. Given a payment amount and the id of a merchant, this algorithm returns a preimage and a signed coin to be spent.
- $\mathsf{CheckPayment}(\mathsf{amount}, \mathsf{x}, \sigma') \rightarrow \mathsf{Ok}/\mathsf{NotOk}$. Given a payment amount, a secret preimage, and a signed coin, this algorithm checks whether or not the payment is correct. Returns Ok if check is successful and NotOk if not.

To make a payment, user Alice reveals the preimage to the merchant. Upon receiving the preimage and the signature σ', the merchant checks with the issuer that the signature σ' is the correct signature for the coin that is obtained from the secret preimage x. In this check process, the merchant informs the issuer of the destination wallet address to be credited with these funds. The issuer then processes a transaction and sends these funds to the corresponding destination wallet address. Upon successful credit of funds, the merchant can release the goods to the customer Alice. This finalizes the protocol.

We refer the reader to Figure 3 for further detail.

3 Security Notes

3.1 Key Compromise

We cover two aspects in this section. First, the key of the issuer can be compromised (i.e., leaked, stolen, ...). Second, the secret preimages of the users that allow for the spending of coins can similarly be stolen or even lost.

Issuer. We propose using $\mathcal{S}_{leeve}$ [9, 7, 8] to ensure that the issuer can always securely 're-key' if needed. Using this construction, the issuer first goes through a slightly more complex key generation process that generates two keys: a fallback key, and a regular ECC key. In this setting, the fallback key is generated securely and kept in a secure 'cold' storage. In the event of a key compromise, we note that the malicious party who gained access the keys is not able to mint money out of thin air. This is the case because a mint is always associated with an input transaction that spends the corresponding funds along with the blinded coin (to be signed). This 1-to-1 mapping ensures that for a mint to take place, the spending of funds must also occur. Therefore, surprisingly, a leak of a secret key of the issuer in this setting does not allow for the minting of any additional money into the circulating supply. This malicious party, however, can generate malicious coins, sign them, and spend them until spending the amount of remaining coins in circulation.

To prevent this attack, upon acknowledging the key compromise, the system can potentially enforce the use of an additional zero-knowledge proof to force the payer to prove (in zero-knowledge) that they know the blinding factor of one of the blinded coins. The adversary would not be able to create such a proof as this implies breaking the discrete log problem (assuming we use ECC blind signatures).

Since we recommend the use of ECC blind signatures in our protocol, this proof can be a composed Chaum-Pedersen discrete log equality (DLEQ) protocol that proves for a set of coins "This coin I am spending is either this coin raised to a blinding factor that I know, OR it is either this coin raised to a blinding factor that I know (and so on)". This proof is linear in work and size so should not be performed for the entire set of blinded coins in the system and only a subset of coins.

Alternatively, there are two other possible approaches. First, since this is dealing with an exceptional case that is of extreme importance, the system can consider breaking the privacy of payments that are taking place during a time period. This ensures that the zero-knowledge proof is constant-sized as the payer must prove the statement for a single blinded coin, thus compromising its privacy. This, however, incentivizes the Issuer to leak the secret key as it can potentially lead to the loss of privacy. Second, this additional zero-knowledge proof can be performed using more modern ZK constructions such as ZK-SNARKs. This potentially reduces the size of the proof and the verification time, but moves the burden of computation mostly to the prover (user making the payment).

We highlight that our construction is able to provide very viable recovery solutions for. what is a doomsday scenario where a big majority of protocols completely collapse. Concretely, the compromise of the secret key of the issuer.

User. Similarly, the user can use the mnemonic to generate the different coins and blinding factors for each coin. If a specific device containing this key material is lost, the user is able to recover all the coins if the mnemonic is stored safely.

3.2 Privacy

An adversary looking at the smart contract, is able to infer that a specific wallet performed a withdrawal operation for a specific amount. Such adversary is also able to obtain the blinded coin associated with that withdrawal. The privacy of the coin, however, is unconditional and the anonymity set of each coin that is spent is equal to the number of users that performed a withdrawal.

We note, for example, that an online payment that includes a delivery may effectively break privacy as the user must insert personal information details such as name and address. Therefore, the identity of the user is private in the sense that the coins spent are theoretically not linked to the user, but then the order must be fulfilled and to ensure that the privacy of the user is compromised.

Censorship. An issuer is able to censor requests from specific merchants if the identity of merchants is disclosed in the payment process. Therefore, upon redemption of funds, merchants should frequently rotate the destination wallet address and potentially the IP address.

4 Discussion

4.1 Comparison with Tornado Cash

Our approach differs from tornado cash as it does not require the use of any type of zero-knowledge proofs to spend funds. These proofs are traditionally expensive for the prover (clients). Our approach does not need the use of relayers to cover gas fees. Our proposal, however, is not as decentralized as Tornado cash

4.2 Escape Hatch

In the event that a user is attempting to perform a withdrawal and the issuer is offline for an extended period of time, the system can enforce additional logic that allows users to make a 'forced' withdrawal and recover their funds. Therefore, users are able to send the funds back to the original wallet address if the issuer is offline during the withdrawal process.

4.3 Asset Support

Our construction supports any type of underlying asset as long as the blockchain supports the 'locking' of assets. We note that, in practice, the issuer should have a key pair for each denomination of coins to be used in the system. Therefore, this is easily extendable to multiple existing blockchains.

4.4 Decentralization of the issuer

The scheme can easily be adapted to have a mode of operation that is distributed (or decentralized). To achieve this extension, the blind signing part, which is effectively an elliptic curve multiplication must be performed in a multiparty computation (MPC) fashion. This makes the use of the quantum-secure fallback quite impractical due to the high-cost of MPC hashing and requires a new distributed key generation (DKG) every time a new participant joins the set of signers.

We note that the issuer can securely generate the signing key and then secret share it amongst a set of trusted parties. Subsequently, if the issuer is offline, these trusted parties can collectively aid the merchants in checking that

4.5 RSA Blind Signatures

In this paper, we recommend the use of elliptic curve blind signatures. This, however, comes with its associated trade-offs. For example, in our proposal, the merchant needs the issuer to confirm that a specific signed coin is valid. This liveness assumption is not ideal as it represents a single point of failure for the processing of payment transactions. Using RSA blind signatures mitigate this issue as anyone is able to check whether a coin is correctly signed or not. These signatures, however, are slower and substantially larger in size.

5 Conclusion

We introduced a novel construction that reduces the trust assumptions from the original e-cash. The security of our construction relies on elliptic curve discrete log problem (ECDLP) instead of prime-factorization (i.e., RSA). This choice is purposefully made to have a construction interoperable with the existing blockchain systems. Our construction relies on well-vetted and well-established cryptographic primitives, is fairly easy to implement, and allows for a clean modular plugin with existing blockchains to provide for cash-like private payments.

References

1. Androulaki, E., Karame, G.O.: Hiding transaction amounts and balances in bitcoin. In: Holz, T., Ioannidis, S. (eds.) Trust and Trustworthy Computing. pp. 161–178. Springer International Publishing, Cham (2014)
2. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: Compact e-cash and simulatable vrfs revisited. Cryptology ePrint Archive, Paper 2009/107 (2009), https://eprint.iacr.org/2009/107, https://eprint.iacr.org/2009/107
3. Brands, S.: Untraceable off-line cash in wallets with observers (extended abstract). In: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology. p. 302–318. CRYPTO '93, Springer-Verlag, Berlin, Heidelberg (1993)
4. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. Cryptology ePrint Archive, Paper 2005/060 (2005), https://eprint.iacr.org/2005/060
5. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology: Proceedings of CRYPTO '82. pp. 199–203. Plenum (1982)
6. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) Advances in Cryptology — CRYPTO' 88. pp. 319–327. Springer New York, New York, NY (1990)
7. Chaum, D., Larangeira, M., Yaksetig, M.: Tweakable sleeve: A novel sleeve construction based on tweakable hash functions. Cryptology ePrint Archive, Paper 2022/888 (2022), https://eprint.iacr.org/2022/888
8. Chaum, D., Larangeira, M., Yaksetig, M.: Wotswana: A generalized sleeve construction for multiple proofs of ownership. Cryptology ePrint Archive, Paper 2022/1623 (2022), https://eprint.iacr.org/2022/1623
9. Chaum, D., Larangeira, M., Yaksetig, M., Carter, W.: W-ots(+) up my sleeve! a hidden secure fallback for cryptocurrency wallets. Cryptology ePrint Archive, Paper 2021/872 (2021), https://eprint.iacr.org/2021/872
10. Lu, Z., Jiang, Z.L., Wu, Y., Wang, X., Zhong, Y.: A lattice-based anonymous distributed e-cash from bitcoin. In: Provable Security: 13th International Conference, ProvSec 2019, Cairns, QLD, Australia, October 1–4, 2019, Proceedings. p. 275–287. Springer-Verlag, Berlin, Heidelberg (2019)
11. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: 2013 IEEE Symposium on Security and Privacy. pp. 397–411 (2013). https://doi.org/10.1109/SP.2013.34

A Withdrawal Protocol

Withdrawal Protocol

To perform a withdrawal of one denominated coin, user Alice performs the following steps:

1. generates a random preimage: $x \xleftarrow{\$} \{0,1\}^n$
2. applies a hash function $\mathcal{H}$ to the preimage to obtain a coin $\mathsf{c} = \mathcal{H}(x)$
3. generates a random blinding factor r
4. computes a blinded coin: $\mathsf{bc} = \mathsf{c}^r \mod p$
5. sends a withdrawal request to the blockchain containing the blinded coin along with a lock transfer for the requested amount.

The smart contract, upon receiving the message, performs the following steps:

1. debits the amount from Alice's account (i.e., wallet).
2. adds amount to balance of the smart contract.

The issuer, upon processing of block containing new withdrawal requests, performs the following steps:

1. signs blinded coin using the correct denomination key
2. issues a zero-knowledge proof π that message is correctly 'signed'
3. posts the signed blinded coin on the blockchain

Alice, upon the processing of the new block, performs the following steps:

1. checks that zero knowledge proof is correct
2. un-blinds the coin
3. signs blinded coin using the correct denomination key
4. stores the newly minted signed coin c^*

Fig. 2. Withdrawal protocol description.

B Payment Protocol

Payment Protocol

To initiate a payment, a customer selects goods they wish to buy and the merchant creates a transaction identifier $\mathsf{tx_{id}}$ for this purchase. The merchant then sends this identifier to customer Alice.

To perform such a payment, Alice performs the following steps:

1. creates a transaction[a] tx by creating a tuple containing the preimage x, the transaction identifier $\mathsf{tx_{id}}$, and a single hash digest of the identifier and the signed coin: $\mathsf{tx} = x,\ \mathsf{tx_{id}},\ \mathcal{H}(\mathsf{tx_{id}}, c^*)$
2. sends the transaction to the merchant.

The merchant validates the payment details and relays the transaction along with the merchant's account information to the issuer.

The issuer, upon receiving the transaction, performs the following steps:

1. applies hash function $\mathcal{H}$ to received preimage x and obtains coin c
2. signs coin c to obtain signed coin c^*
3. checks for double spending by checking in the storage layer (blockchain) if coin c is in the list of allowed coins $\mathcal{C}$
4. produces a hash digest of the transaction identifier and the signed coin: $\mathcal{H}(\mathsf{tx_{id}}, c^*)$
5. checks if the obtained hash digest from previous step matches the hash digest received in the transaction relayed by the merchant.
6. returns an approval on the transaction if all the above checks are successful and credits the account of the merchant.

The smart contract, upon receiving the transaction from the issuer, performs the following steps:

1. checks for double spending by checking in the storage layer if coin c is in the list of allowed coins $\mathcal{C}$
2. returns an approval on the transaction if all the above checks are successful and credits the account of the merchant.

The merchant releases the product(s) to the customer.

[a] Alternatively, the system can use hash-based signatures, where Alice signs the payee and tx details. The merchant can then check that the signature verifies correctly under the public-key (coin), which is signed by the Issuer.

Fig. 3. Payment protocol description.