SπPETs: Sustainable Practically Indistinguishable Privacy-Enhanced Transactions

Timofey Yaluhin
timofey@chainsafe.io
ChainSafe Systems

January 7, 2023

Abstract

A pursuit of chain-layer privacy had spawned many privacy-enhancing technologies such as coin-mixing services, private ledgers, stealth addresses, etc. While achieving anonymity and confidentiality for various aspects of on-chain activity, these solutions leave explicit traces that can be used by powerful actors to push bans and threaten the fungibility of an underlying asset. This work describes a way to achieve covert privacy-enhanced transactions on any public blockchain that supports ECDSA or Schnorr. Our method applies two-party computation combined with adaptor signatures and verifiable timed commitments. We also analyze some of the possible use cases on Ethereum and provide prototype implementation written in Rust.

1. Introduction

The application-level attempts to provide privacy on Ethereum L1 have proved to be vulnerable to state-backed interventions and censorship. While the continuous attempts to ban privacy-preserving technologies may not discourage regular users from seeking anonymity and confidentiality for their on-chain activity, they did threaten the fungibility of the underlying assets, which is arguably the most important property to protect.

In our previous work [1], we proposed using multi-party computation to trustlessly mix coins in a shared account without a direct association with an on-chain smart contract. Although such construction could theoretically provide stronger censorship resistance, its security was questionable and hardly achievable in practice. Furthermore, it was still fairly possible to tell that offchain mixing had taken place by doing chain analysis, so the fungibility guarantee could still be undermined with enough cooperation.

It has soon become apparent that truly sustainable privacy in the context of public (not privacy-oriented) blockchains can only be achieved if privacy-enhanced transactions are indistinguishable from regular ones. This is the core idea of the CoinSwap protocol proposed by Greg Maxwell in 2013 and revisited by Chris Belcher in a more recent paper [2].

In this work, we introduce a covert privacy-enhanced transactions (PETs) protocol, which is universally applicable to all public blockchains that support ECDSA or Schnorr. In the rest of the paper, we briefly describe the CoinSwap protocol and some additional techniques needed to practically deploy it on a public blockchain network. We then introduce cryptographic primitives that we use as building blocks for out construction. In Section 4, we specify our protocol along with some extensions (delayed withdrawals to mitigate time correlation). In Section 5 we go over some practical consideration and possible use case for deploying our protocol to the Ethereum mainnet. Complementary to this paper, we also developed a prototype[1] to evaluate and validate concepts described in this work.

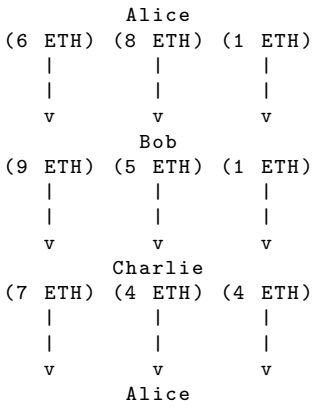
2. Reference materials

2.1 CoinSwap

CoinSwap is a non-custodial way of swapping one coin for another one. Alice can swap coins with Bob by first sending them to a CoinSwap address and having those coins then sent to Bob:

[1] https://github.com/timoth-y/spy-pets

Figure 1: Combining multi-transaction with routing



Alice's Address 1 ——> CoinSwap Address 1 ——> Bob's Address 1

An entirely separate set of transactions gives Bob's coins to Alice in return:

Bob's Address 2 ——> CoinSwap Address 2 ——> Alice's Address 2

Privacy is improved because on-chain there is no explicit link between `Alice's Address 1` and `Alice's Address 2`. All that is seen is a regular transfer to Bob.

Moreover, anyone analyzing the chain must now deal with the possibility that any transfer transaction is actually a covert PET sending coins to a totally unconnected address. So overall privacy is improved for all users even without coordinated changes in behavior.

However, due to the amount and time correlation, CoinSwap alone won't be enough to provide any significant privacy boost. It is also vulnerable to active adversaries, who by taking Bob's role can undermine the anonymity that Alice had just gained. Multi-transactions and multi-hop routing can be used on top of CoinSwap to beat both of these threats respectively.

With multi-transactions, Alice splits the desired amount into multiple transactions, this way she gets back multiple deposits that add up to the same amount, but nowhere on the blockchain is a record that Alice received exactly the same as Bob had.

With multi-hop routing, Alice routes her transactions through many Bobs, such that it would be impossible for anyone of them to tell whether the incoming coins came from Alice herself or the previous Bob in her route. Only Alice will know the entire route, while the intermediaries will only know the previous and next blockchain addresses along the route.

We will not go into more detail here, but we strongly encourage readers to check Chris Belcher's design paper [2] for a much more thorough overview.

In the following sections, we introduce techniques and cryptographic primitives that would help us implement the CoinSwap protocol on any blockchain without using on-chain scripts of any kind.

3. Building blocks

3.1 2-Party ECDSA

The original CoinSwap idea uses 2-of-2 multi-sig. While this can easily be achieved using smart contracts, this way the covertness will be canceled out, as now all the PETs could be traced back to the contract they rely on. This would make sense for smart contract wallets, though their combined anonymity set would still be much weaker than that of EOAs.

We would instead use a 2-party computation (2PC) to jointly produce a valid ECDSA signature for a transaction that claims previously locked ETH from CoinSwap address. Namely, the protocol that we choose for this is *Lindell'17* [3]. It's a widely used, battle-tested, and virtually state-of-the-art implementation of 2P-ECDSA.

Following this discovery, the natural intuition now would be to go ahead and use a secure 2PC protocol to compute two signatures, one for each swap. However, this naive attempt leads to an insecure scheme: the 2PC protocol does not guarantee[2] output delivery, so nothing prevents Bob from going offline after receiving a valid signature for his transaction.

[2]It may be tempting to run two instances of 2P-ECDSA in parallel, so that on each round parties exchange both sets of intermediary values. This, however, solves nothing since any party can withhold the last intermediary value on the last round.

3.2 Adaptor Signatures

There exist another cryptographic primitive that is suited to solve such fairness problems — adaptor signatures [4] AKA pre-signatures AKA one-time verifiably-encrypted signatures (VES).

This fascinating construction was invented by Andrew Poelstra and was quickly adopted by the Bitcoin community as a means of achieving "scriptless scripts" - a way of encoding rules by arranging cryptography primitives rather than relying on high-level scripting languages. We refer the interested reader to check Lloyd Fournier's survey paper [5] on the subject.

Adaptor signatures are a special kind of signature encryption. When an encrypted signature (adaptor) - owned by Alice - is combined with a decrypted signature - revealed by Bob - they together yield a decryption key (witness).

One can use this weird leakage of the decryption key to constructing an atomic swap protocol. As soon as Alice publishes a claim transaction with a decrypted signature attached, Bob is able to download it and use it to recover a key corresponding to the wallet where his swapped coins are locked.

This construction works well for UTXO chains like Bitcoin but has a fundamental flaw on account-based blockchains like Ethereum. Namely, Bob can front-run Alice's attempt to send a signed transaction by publishing a transaction with the same nonce shortly before. If Alice misses noticing that and publishes the compromised transaction anyway, Bob can take a decrypted signature from the mempool and claim coins unfairly.

3.3 Joint Adaptor Signing

Fortunately, we can solve both problems with introduced primitives by combing them together into a brand new *2P-AdaptorECDSA* scheme. The shared EOA component of *Lindell'17* prevents front-running attacks, as there's no way to sign a conflicting transaction without a mutual agreement from both sides.

Simultaneously, adaptor signatures guarantee the fairness of the exchange, ie. by claiming from `CoinSwap Address 1` Alice releases the decryption key y that Bob can use to claim from `CoinSwap Address 2`. More specifically, during the "Lock" phase we require both parties to jointly generate two adaptor signatures over the same witness y. This way, Alice who initially knows y will publish her transaction first, thereby allowing Bob to extract y and use it to adapt his signature and publish next.

Constructing a protocol with adaptor signatures has the additional benefit of efficiently scaling with an increase of transactions to be signed. This is useful for multi-transactions: imagine Alice decides to route her `5 ETH` transfer using Bob(A) `3 ETH` and Bob(B) `2 ETH`. As long as pre-signatures for all transactions are generated with respect to the same instance Y, Bob would only need to publish once (`5 ETH` $\rightarrow$ Bob(A)) and Alice could then adapt both her transaction.

Malavolta et al. described *2P-AdaptorECDSA* scheme in their work [6] It was later applied in the context of universal atomic swap in *TMM'21* [7] paper, which was of great inspiration for this protocol.

3.4 Verifiable Timed Commitments

As we fixed the problem with fairness, we, unfortunately, introduced another one with fault tolerance. In a case where Alice deposits to `CoinSwap Address 1` and Bob goes offline before the pre-signature is generated, Alice's coins will be forever locked. The standard way of addressing this is by using a hash-time lock contract (HTLC) that allows user to refund their coins after a certain timeout. Fallback to on-chain scripts will obviously ruin the privacy gains we've established up until this point. The solution here must be scriptless too.

Likely, we can simulate such functionality using verifiable timed commitments (VTC), which lets a user (committer) to generate a timed commitment C of some witness x. The commitment C must hide witness x for time $\mathbf{T}$ (which can be chosen arbitrarily by the committer). At the same time, the committer also generates a proof π that proves that the commitment C contains a *valid* witness x. This guarantees that x can be publicly recovered in time $\mathbf{T}$ by anyone who solves the computational puzzle.

The notion of *validity* will vary depending on what role the witness plays in the protocol. It's entirely possible to have a verifiable timed signature (VTS) where x is a valid signature for a refund transaction. This obvious choice, however, would come with the cost of the expensive setup (7 seconds) and verification (10 seconds). Instead, the committer can prove that the witness is a valid

discrete logarithm (VTD) of some known element, which is a much simpler algebraic statement and only takes a few milliseconds to setup and verify.

The idea is for Bob to time-lock his key share $x_{(b)}$ of the `CoinSwap Address 1` inside a VTD. Upon receiving Alice could use proof π to verify that $G^{x_{(b)}} \cdot G^{x_{(a)}} = X$ where instances $x_{(a)}$ (key share) and X (public key) is known to her.

3.4.1 Homomorphic Time-Lock Puzzles

The construction of VTD from [8] makes use of the homomorphic time-lock puzzles (HTLP) [9], a primitive that lets one to encapsulate a secret inside a puzzle with the guarantee that the puzzle can be opened only after time **T**. As the name implies it supports certain homomorphic properties. The authors use this additive homomorphism and cut-and-choose proving technique to construct an efficient NIZK proof for a statement $H = G^x$.

The biggest challenge with time-lock puzzles is about choosing the Timing Hardness parameters **T**. While HTLP guarantees that users with a large number of CPU cores won't be able to solve a puzzle quicker than if they only have one, the speed of individual CPU cores does make a difference. In [7] authors suggest that parties shall make conservative estimates of each others computational power in opening the VTD commitments. In reality, however, there's no reliable way to exchange such information, so parties should base their estimates on the most performant hardware available on the market.

Since the claim timeout should only be a few minutes, we are generally fine if for some users opening of VTD would take double that. Furthermore, in practice, we expect that the presence of VTD will mostly function as a deterrent for users not to misbehave and the parties will not have to open them, except for rare cases.

3.4.2 Distributed Time-Lock Systems

Similarly how HTLC prevents coins to be released until a certain block height is mined, one can rely on some untrusted third party to release the decryption key when a certain time is reached. A promising technology we can use for such a purpose is — the *"tlock"*[3] distributed timelock encryption system.

In the *tlock* system an untrusted party is replaced with the Drand threshold network and messages are locked using the identity-based encryption (IBE) [10] - a scheme where public key can be an arbitrary string.

The Drand network generates randomness in rounds at specific time intervals (30 seconds on the mainnet). Anyone can time-lock a message by encrypting it with a public key, derived from an arbitrarily chosen round number. When the randomness associated with this round is released, it in effect releases the private key associated with the round's identity public key. Note, that this scheme is non-interactive, so the committer doesnt need to participate in the decryption process.

By itself, *tlock* encryption is just that - an elaborate encryption scheme. It does not guarantee anything about the plaintext validity, so one can still cheat by encrypting some random bytes and just pretend that they have used proper key material.

3.4.3 zk-TimeLock

To prevent cheating, we propose an extended *zk-timelock* scheme that comes with programmable witness verifiability. For this, we implement IBE inside an arithmetic circuit that checks the aforementioned discrete log relation[4].

The algorithm we are adapting is *Boneh-Franklin's* [10] IBE scheme. The primary source of constraint blow-up is in the target group (pairing product) operations, specifically `Gt` $\cdot$ `Fr` multiplication. All operations must also be projected on top of the BLS12-381 curve, one that Drand operates on. This poses an even bigger problem as there is no commonly known[5] pairing-friendly curve whose scalar field equals the base field of BLS12-381, which is needed for efficient KZG-based SNARKs. When looking for a solution, we have conducted a set of experiments with different circuit configurations and arrived at two that appear reasonably practical.

[3] https://github.com/drand/tlock

[4] In practice the expressivity of the arithmetization used in the underlying proving system could allow asserting relations that are much more complex than just dlog proof: one can check the validity of an ECDSA signature using a public key as a public instance.

[5] The reason why such a popular curve as BLS12-381 doesn't have an equally known embedding curve lies in the properties of its base field. Namely, its Fq's two-adicity is low (1), which contributes to the embedding curve having proportionally low FFT space, making them unusable with conventional proving systems like Groth'16.

The first one is (1) a control case circuit with native arithmetic only that can be projected on any elliptic curve. We tested it using the Groth'16 proving system with BLS12-377/BW6-671 curve combination. While this configuration was originally intended for comparison with other methods, it unsparingly remains the most efficient. Because of this, we have submitted a proposal to extend the Drand protocol with BLS12-377 curve support.

For the second configuration (2) we introduce *YT6-776* — an application-specific curve that embeds BLS12-381's base field. The name denotes that it was generated using the Cocks-Pinch method for the embedding degree 6. The size of the curve is 776 bits, hence its performance is comparable BW6-761 curve that embeds BLS12-377. We use YT6-776 in combination with Gemini [11] — an elastic proving system that is FFT-free, more curve-agnostic, and hardware-friendly.

For both configurations we get the results in the same order of magnitude[6], concluding that practical verifiable time-lock encryption using state-of-the-art zero-knowledge techniques can be achieved today already.

4. SπPETs: A Covert-PETs protocol

In this section, we present our covert PETs protocol build using the cryptographic building blocks we have introduced so far.

We assume a public bulletin board where users (market-takers) can find relayers (market-makers). In the following notation, Bob is a market-maker, he runs a relayer server and advertises its address on a bulletin board. Alice wishes to swap her coins, thereby she is a market-taker. She finds a relayer that has enough liquidity for the amount she wishes to swap. For choosing this amount Alice can pay Bob a fee.

PROTOCOL 4.1: Universal Covert PETs (happy path/part 1)

Initialize: Alice generate a new destination address D_a, where swapped coins will go. Bob can choose to generate a new destination address D_b or use an existing one.

Parties agree on the amount α, time parameters t_a, t_b and work as follows:

1. **Alice's first message (`Taker::Setup`):**
   (a) Alice takes P2's role in [3] and computes two `KeyGen::P2::Msg1`, one for each CoinSwap account (S_1, S_2) and sends it to Bob.
2. **Bob's first message (`Maker::Setup`):**
   (a) Bob takes P1's role in [3] and computes `KeyGen::P1::Msg1` for S_1, S_2. Using the tuple of `KeyGen::P2::Msg1` received from Alice, he also computes `KeyGen::P1::Msg2` for S_1, S_2.
   (b) Bob sends tuple of `KeyGen::P1::Msg1`, tuple of `KeyGen::P1::Msg2` to Alice.
3. **Alice's second message (`Taker::Lock`):**
   (a) Using the both sets of `KeyGen::P1::Msg1` and `KeyGen::P1::Msg2` received from Bob, Alice computes `KeyGen::P2::Msg2` for S_1, S_2.
   (b) Alice chooses a random scalar y and computes `Sign::P2::Msg1` according to [6] and sends it to Bob.
4. **Bob's second message (`Maker::Lock`):**
   (a) Bob computes hash h_b of the transaction that transfer α coins from S_1 into D_b.
   (b) Bob encloses for time t_a his local key share $x_{p_1}^{S_1}$ into commitment C_a and proof π_a according to VTC scheme.
   (c) Bob compute `Sign::P1::Msg1` according to [6] and sends it to Alice along with C_a, π_a and h_b.

[6] Full benchmarks along with a reference implementation source code are available in the dedicated GitHub repository: https://github.com/timoth-y/zk-timelock

PROTOCOL 4.1: Universal Covert PETs (happy path/part 2)

5. **Alice's third message (`Taker::Swap`):**
   (a) Alice verifies proof π_a. If VTC is an HTLP scheme, Alice also starts solving puzzles to unlock C_a received from Bob.
   (b) Alice deposits α coins to S_1 and computes hash h_a of the transaction that transfer α coins from S_2 to D_a.
   (c) Alice encloses for time t_b her local key share $x_{p_2}^{S_1}$ into commitment C_b and proof π_b.
   (d) Using local shares $x_a^{S_1}$ and $x_a^{S_2}$ Alice computes `Sign::P2::Msg2` for h_b, h_a respectively and sends them to Bob along with C_b, π_b.
6. **Bob's third message (`Maker::Swap`):**
   (a) Bob verifies dlog proofs from `Sign::P2::Msg2` according to commitments received earlier as a part of `Sign::P2::Msg1`.
   (b) Bob verifies proof π_b. If VTC is an HTLP scheme, Bob starts solving puzzles to unlock C_b.
   (c) Bob deposits α coins to S_2. Using local shares $x_b^{S_1}$ and $x_b^{S_2}$ he computes σ'_b, σ'_a (pre-signatures) for h_b, h_a respectively.
   (d) Bob sends σ'_a to Alice and listens to chain events for Alice to broadcast a valid signature σ_a.
7. **Output:**
   (a) Alice decrypts σ'_a using key y to get a valid signature σ_a, which she then broadcasts on-chain along with the transaction that transfers α coins from S_2 to D_a.
   (b) Bob downloads σ_a to recover witness y from σ'_a and decrypts σ'_b using key y to get a valid signature σ_b, which he then broadcasts on-chain along with the transaction that transfers α coins from S_1 to D_b.

PROTOCOL 4.2: Universal Covert PETs (refund paths)

Bob goes offline after Alice deposits in step 5:

(a) After time t_a Alice opens timed commitment C_a and acquires Bob's key share $x_{p_1}^{S_1}$, which she then multiple by the local $x_{p_2}^{S_1}$ to get valid key x^{S_1}.

(b) Having x^{S_1}, Alice can now sign the transaction to transfer α from S_1 to other account of her choice.

(b) When Bob is back online and time t_b has passed, he opens timed commitment C_b and multiples its opening on $x_{p_1}^{S_2}$ to get valid key x^{S_2}.

(b) Having x^{S_2}, Bob can now sign the transaction to transfer α from S_2 to other account of his choice.

Alice goes offline before broadcasting a valid signature in step 7:

(a) After time t_b Bob opens timed commitment C_b and acquires Alice's key share $x_{p2}^{S_2}$, which he then multiples by the local $x_{p_1}^{S_2}$ to get valid key x^{S_2}.

(b) Having x^{S_2}, Bob can now sign the transaction to transfer α from S_2 to other account of his choice.

(b) When Alice is back online and time t_a has passed, she opens timed commitment C_a and multiples its opening on $x_{p_2}^{S_1}$ to get valid key x^{S_1}.

(b) Having x^{S_1}, Alice can now sign the transaction to transfer α from S_1 to other account of her choice.

4.1 Delayed withdrawals

The CoinSwap design proposed in [2] covers amount-correlation attacks, but has no additional defense from time-correlation. On the other hand, the current generation of contract-based privacy solutions, such as TornadoCash come with the ability to delay withdrawals for an arbitrary time. We can support similar functionality using the VTC primitive that we introduced in Section 3.4.

Alice can ask Bob to delay his withdrawal from CoinSwap account for some arbitrary time t_Δ. To enforce this she must enclose a certain value, which Bob needs to complete his withdrawal, into a VTC that will hide for a time t_Δ. The most obvious candidate is an adaptor of Bob's withdrawal transaction signature. However, in our protocol Bob is the one who generates all adaptors, so instead we propose time-locking r_x scalar from [6] needed to decrypt adaptor along with a key recovered by combining valid signature and its corresponding adaptor in Step 7. The r_x witness is also much simpler to generate proof for, compared to an adaptor or something else entirely. This is because r_x is a regular 32-byte scalar that is revealed along with R_x point such that $R_x = r_x G$ which can be proved using trivial discreet log proofs.

The resulting construction assumes no more precision in terms of opening time than the refund mechanism does, so both HTLP and zk-TimeLock versions can be applied. However, since Alice may want to delay withdrawals on a possible long duration e.g. several hours, use of HTLP may end up being problematic: the longer the delay, the more work Bob needs to perform to open VTC and proceed to withdraw his funds. On the other hand, work associated with opening commitments generated using zk-TimeLock is constant and independent from t_Δ. Hence, we recommend using this method for implementing delayed withdrawals.

5. Ethereum deployment

5.1 Use cases

On the Ethereum blockchain, users pay for requested computation in gas using ether cryptocurrency. Determining the source of funds deposited to pay these network fees is essentially the way of de-anonymizing users on public pseudo-anonymous networks. Hence, the aim of privacy-enhanced transactions in the context of functional blockchains — is to hide who and when enters and leaves certain dApps, i.e. **privacy-enhanced on/off-ramping.**

In the original CoinSwap idea, shared addresses are merely seen as an intermediary pit stop where coins are locked for parties to agree on their further destination. In the context of functional blockchains, we can extend their role to support **arbitrary contract invocation**. Once `CoinSwap Address 2` is funded by Bob, parties can use *2P-ECDSA* to sign any type of transaction requested by Alice. Again, on-chain there is no direct link that connects Alice to that transaction, all that is seen is Bob interacting with some dApp from his "new" address.

We identify several types of Ethereum applications where such an exploit would make sense: 1) exchange coins on the decentralized exchange (DEX) platform, 2) send coins to another network using a cross-chain bridge, 3) deposit coins into a pool in exchange form liquidity-provider (LP) tokens. This underpins another relevant functionality that our protocol is able to support, that being — **privacy-enhanced ERC20 operations.** Note, that it would still require someone to pay for gas, so the number of transactions to be signed will equal 1 + (whatever is needed to transact with a given ERC20 token).

5.2 Cost

A reasonable concern with having the CoinSwap protocol on Ethereum is about the economic cost of such privacy gains. The simplest form of CoinSwap transaction would cost double what the traditional transfer does ($2 * 21000 = 42000$ gas). An interaction that includes 3 input multi-transaction and 3 hops routing is expected to provide an excellent privacy guarantee for a cost of $21000 * 18 = 378000$ gas, which is about the same cost as withdrawing from the Tornado Cash pool. However, such complex multi-transaction routed coin-swaps are only for the highest threat models where the makers themselves are adversaries. In practice, most users would probably choose to use just one or two hops.

Deploying CoinSwap protocol on Ethereum L2s AKA Rollups will result in multiple CoinSwap transactions being sequenced and batched together. This will reduce operational costs while preserving unique privacy and indistinguishability properties. Users could then leverage the cheapest L2 which supports CoinSwap for anonymously on-ramping into the Ethereum ecosystem and then use the deposited funds elsewhere.

5.3 Implementation

In order to show the feasibility of our construction, we implemented it and hereby release our reference implementation, called SpyPETs, and its source code to the cryptocurrency community[7].

6. Conclusion

We have presented a way of implementing a Privacy-Enhanced Transactions protocol similar to Bitcoin's CoinSwaps on any blockchain that supports ECDSA or Schnorr. Our construction assumes no scripting/smart-contract capabilities of the hosting chain. Instead, a special combination of cryptographic primitives *(2P-AdaptorECDSA + Verifiable Timed Commitments)* is used to allow parties to engage in the atomic-swap-like protocol in a scriptless and offchain manner.

Apart from being *universally compatible* with a wide variety of blockchains, including scriptless ones such as Stellar and Ripple, our protocol is also *covert* — meaning that the resulting transactions are virtually indistinguishable from any other coin-transfer transaction on the ledger. The latter is especially valuable as it underpins the sustainability of privacy-enhancing technology and the fungibility of assets it's applied to.

All in all, Covert PETs have a strong premise to provide anonymity to those who use them. Moreover, by virtue of being indistinguishable from other on-chain interactions, it can also improve privacy for those who do not and perhaps never even heard about this protocol.

7. Acknowledgments

I would like to thank István András Seres for introducing me to the CoinSwap protocol. I also thank Elizabeth Binks and István András Seres for insightful discussions and their valuable feedback on earlier drafts of this paper and reference implementation. Finally, I thank Weikeng Chen for sharing the method of creating application-specific curves described in [12] and for helpful discussions about it.

References

[1] E. Binks and T. Yaluhin, "Offchain and scriptless mixer." https://hackmd.io/@nZ-twauPRISEa6G9zg3XRw/rygZ6fvrc, 2022.

[2] C. Belcher, "Design for a coinswap implementation for massively improving bitcoin privacy and fungibility." https://gist.github.com/chris-belcher/9144bd57a91c194e332fb5ca371d0964, 2020.

[3] Y. Lindell, "Fast secure two-party ecdsa signing," 2017.

[4] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized channels from limited blockchain scripts and adaptor signatures." Cryptology ePrint Archive, Paper 2020/476, 2020. https://eprint.iacr.org/2020/476.

[5] L. Fournier, "One-time veriably encrypted signatures a.k.a. adaptor signatures." https://github.com/LLFourn/one-time-VES/blob/master/main.pdf, 2019.

[6] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," 2018.

[7] G. Malavolta, S. A. K. Thyagarajan, and P. Moreno-Snchez, "Universal atomic swaps: Secure exchange of coins across all blockchains,"

[8] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Dttling, A. Kate, and D. Schrder, "Verifiable timed signatures made practical," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, (Virtual Event USA), p. 17331750, ACM, Oct 2020.

[9] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications,"

[10] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," p. 31, 2001.

[7] https://github.com/timoth-y/spy-pets

[11] J. Bootle, A. Chiesa, Y. Hu, and M. Orr, "Gemini: Elastic snarks for diverse environments," 2022. Report Number: 420.

[12] W. Chen and R. Akeela, "Yafa-108/146: Implementing ed25519-embedding cocks-pinch curves in arkworks-rs,"