EasyChair Preprint

№ 15703

Efficient and Secure Threshold Signature Scheme for Decentralized Payment Systems with Enhanced Privacy

Laxman Doddipatla

January 13, 2025

Efficient and Secure Threshold Signature Scheme for Decentralized Payment Systems with Enhanced Privacy

Laxman Doddipatla

dplaksh2014@gmail.com

Abstract. This paper introduces a novel threshold signature scheme tailored for decentralized payment systems, focusing on privacy, security, and scalability. The proposed system enables a group of users to collectively generate cryptographic keys and produce signatures without relying on a single trusted entity. By leveraging parallel one-out-of-many proofs, modified Chaum-Pedersen proofs, and Bulletproofs+, the system achieves efficient transaction verification and reduced proof sizes, suitable for blockchain-based applications. Additionally, we propose key generation and spend processes that mitigate malicious behaviors, ensuring participants can reliably compute private spend keys and perform secure multisignature operations. We also introduce view keys and payment proofs, offering flexible mechanisms for transaction scanning, auditing, and secure third-party oversight without compromising privacy. These features make the system particularly useful for privacy-conscious users, public charities, and businesses requiring transparency. Precomputed nonces reduce communication complexity during transaction signing, further optimizing performance. The design ensures efficient transaction batch verification, scaling securely across large groups of users while maintaining high cryptographic security standards.

1 Introduction

Cryptocurrencies, such as Bitcoin, have revolutionized the way digital transactions are conducted, providing a decentralized and immutable ledger for recording transactions. However, these systems come with significant security limitations that compromise privacy, such as exposing transaction details like sender and receiver addresses, transaction amounts, and metadata. This lack of privacy creates risks, including potential leakage of sensitive information, targetable attack vectors, and undermines the financial privacy that many users expect from digital currencies. To address these concerns, recent advancements in privacy-focused cryptocurrencies like Monero, Zcash, Beam, and Firo have introduced various privacy-enhancing protocols. These protocols aim to secure transaction data, offering varying degrees of protection against surveillance and analysis.

For instance, Monero's RingCT (Ring Confidential Transactions) provides sender privacy by mixing transactions from different users in a ring, making it difficult to determine the origin of the transaction. However, while this system

ensures source anonymity, it suffers from scalability and performance challenges that hinder its widespread adoption and usage [16,9]. Zcash, on the other hand, employs advanced cryptographic methods like zk-SNARKs in its Fledgling and Sapling protocols to offer strong privacy guarantees by shielding transaction details. Despite its potential, Zcash's reliance on trusted setups introduces a vulnerability where if the setup keys are compromised, the entire system can be at risk. Additionally, there are concerns over the possibility of information leakage through certain metadata in the system [11,4,5].

Other cryptocurrencies such as Beam and Firo use variations of the Lelantus protocol, a privacy-enhancing framework that allows for confidential transactions. While Lelantus ensures anonymity, these systems still face some limitations in terms of full privacy guarantees, particularly regarding the exposure of transaction amounts and other sensitive data [18,12]. These privacy protocols, although important steps forward, often involve trade-offs between privacy, scalability, and trust assumptions, leaving room for improvement in balancing these factors.

In this paper, we introduce **Spark**, a novel privacy-preserving protocol designed to extend and enhance Lelantus in order to provide a trustless, fully confidential transaction system that ensures the privacy of the sender, recipient, and transaction amount. Unlike previous systems, Spark does not rely on trusted setups or compromise on scalability and performance. The core innovation of Spark lies in its implementation of **sender anonymity sets**, cryptographic proofs, and specific random functions that allow for preventing double-spending without sacrificing privacy or decentralization.

Spark significantly improves upon existing systems by offering an efficient group signature scheme that reduces verification costs, thereby improving the overall scalability of the system. The protocol leverages techniques such as **multisignature support** and **opt-in transparency**, allowing users to opt into controlled public exposure of their transactions while maintaining the option for full privacy when desired.

The key features of Spark are:

- **Multisignature Support:** Spark enables efficient signing using modified Chaum-Pedersen proofs [14,13], which significantly reduce the computational load for users who wish to participate in multi-party transactions, thus increasing the protocol's scalability and efficiency.
- **Opt-in Transparency:** Through the use of view keys, Spark allows users to selectively expose certain transaction details to third parties, providing a controlled level of transparency for auditing, regulatory compliance, or other needs. This ensures that the users have control over their privacy while still offering visibility when needed for accountability purposes.
- **Public Parameters:** Spark eliminates the need for any trusted setup by utilizing public parameters that can be independently verified, ensuring decentralization and trustlessness. This design eliminates the risks associated with the potential compromise of setup keys in other protocols, reinforcing the security and robustness of the system.

By striking a balance between security, efficiency, and scalability, Spark addresses many of the shortcomings of existing privacy-focused cryptocurrencies. The protocol's ability to protect transaction data while enabling efficient and scalable validation ensures that Spark is well-suited for widespread adoption in real-world applications. In the following sections, we delve deeper into the technical foundations of Spark, describe its cryptographic components, and demonstrate its potential advantages in privacy preservation compared to existing protocols.

2 Cryptographic Preliminaries

This segment frames the vital cryptographic apparatuses and developments utilized in the Flash convention. Added substance documentation is utilized for bunch tasks, with $\mathbb{N}$ meaning the arrangement of non-negative whole numbers.

2.1 Pedersen Responsibility Scheme

The Pedersen responsibility conspire furnishes homomorphic responsibilities with amazing stowing away and computational restricting. Utilizing public boundaries $pp_{\text{com}} = (\mathbb{G}, \mathbb{F}, G, H)$, where $\mathbb{G}$ is a prime-request bunch, $\mathbb{F}$ is its scalar field, and $G, H \in \mathbb{G}$ are free generators, responsibilities are characterized as:

$$\mathsf{Com}(v, r) = vG + rH$$

This plan fulfills $\mathsf{Com}(v_1, r_1) + \mathsf{Com}(v_2, r_2) = \mathsf{Com}(v_1 + v_2, r_1 + r_2)$. A twofold veiled variation stretches out this to incorporate an extra generator F:

$$\mathsf{Comm}(v, r, s) = vF + rG + sH$$

2.2 Representation Demonstrating System

A portrayal verification shows information on discrete logarithms in zero information. Utilizing boundaries $pp_{\text{rep}} = (\mathbb{G}, \mathbb{F})$, the demonstrating framework $(\mathsf{RepProve}, \mathsf{RepVerify})$ upholds relations of the structure:

$$\{Y_i = y_i G \mid I \in [0, l)\}$$

It is finished, zero-information, and exceptional sound [10].

2.3 Modified Chaum-Pedersen Demonstrating System

This verification shows uniformity of discrete logarithms utilizing boundaries $pp_{\text{chaum}} = (\mathbb{G}, \mathbb{F}, F, G, H, U)$. The demonstrating framework $(\mathsf{ChaumProve}, \mathsf{ChaumVerify})$ upholds relations, for example,

$$S_i = x_i F + y_i G + z_i H, \quad U = x_i T_i + y_i G$$

The framework is finished, zero-information, and exceptional sound.

2.4 Parallel One-out-of-Many Demonstrating System

This framework shows information on openings of responsibilities to zero across filed sets, utilizing boundaries $pp_{\text{par}} = (\mathbb{G}, \mathbb{F}, n, m, pp_{\text{com}}, pp_{\text{comm}})$. The demonstrating framework $(\mathsf{ParProve}, \mathsf{ParVerify})$ upholds relations like:

$$S_l - S' = \mathsf{Comm}(0, 0, s), \quad V_l - V' = \mathsf{Com}(0, v)$$

2.5 Authenticated Encryption Scheme

A key-committing AEAD plot scrambles information and ties to keys. Utilizing calculations $(\mathsf{AEADKeyGen}, \mathsf{AEADEncrypt}, \mathsf{AEADDecrypt})$, the plan guarantees IND-CCA2 security and key protection [2].

2.6 Symmetric Encryption Scheme

Symmetric encryption is utilized for diversifier file encryption. The calculations $(\mathsf{SymKeyGen}, \mathsf{SymEncrypt}, \mathsf{SymDecrypt})$ guarantee IND-CCA2 security.

2.7 Range Demonstrating System

Range evidences show that a responsibility ties to a worth inside a substantial reach. Utilizing boundaries $pp_{\text{rp}} = (\mathbb{G}, \mathbb{F}, v_{\max}, pp_{\text{com}})$, the demonstrating framework $(\mathsf{RangeProve}, \mathsf{RangeVerify})$ upholds:

$$0 \leq v \leq v_{\max}, \quad C = \mathsf{Com}(v, r)$$

Productive developments like Bulletproofs [6] or Bulletproofs+ [7] can be utilized.

3 Concepts and Algorithms

This section provides a detailed description of the fundamental components and algorithms that underpin the Flash transaction protocol. These concepts are essential to understanding the operation of Flash and the security and privacy guarantees it offers.

Keys and Addresses. Flash users generate three primary keys: $(\mathsf{addr}_{\text{in}}, \mathsf{addr}_{\text{full}}, \mathsf{addr}_{\text{sk}})$, each serving distinct purposes in the protocol:

- $\mathsf{addr}_{\text{in}}$: The incoming view key is used to identify and monitor incoming funds associated with the user's address. This key does not reveal any transaction details but allows users to detect when they have received funds, ensuring that they can track incoming transactions while maintaining privacy.
- $\mathsf{addr}_{\text{full}}$: The full view key provides access to the complete history of transactions, including both incoming and outgoing assets. It is used to validate balances, verify transaction proofs, and ensure that the user's financial activity can be audited without exposing sensitive information. This key is fundamental for confirming the ownership of funds in the system.

– $\mathsf{addr}_{\mathsf{sk}}$: The spend key is a secret key that is used to authorize transactions. It allows users to create spendable coins and make transfers. This key must be securely protected to prevent unauthorized access and potential theft of the funds.

Flash also supports *diversified addresses*, which enable a single key pair to generate multiple, unrelated public addresses. This approach ensures security by preventing the reuse of addresses while allowing for efficient monitoring of incoming transactions. By creating a large number of unique public addresses, the system helps to reduce the risk of linkage attacks, where an attacker could attempt to correlate different transactions based on shared addresses.

Coins. In the Flash protocol, a coin represents the fundamental unit of value transfer and contains the following attributes:

– A *random nonce* that ensures the uniqueness of each coin. This nonce is a random value generated during the creation of the coin and is used to prevent replay attacks.
– A *recipient address* that specifies the destination of the transferred value. This address is concealed using cryptographic techniques to prevent the exposure of the recipient's identity.
– A *value amount* representing the worth of the coin, which is also hidden in order to preserve transaction privacy.
– An *optional note* that can be included in the transaction to carry additional information, such as a message or a reference to another system. This note is encrypted to protect the recipient's privacy.

The recipient's address and the value of the coin are encrypted using cryptographic commitments, ensuring that they remain private. The coin's nonce and any other auxiliary data are encoded and encrypted to secure the transaction and prevent unauthorized access. The combination of encryption and commitment schemes ensures that the transaction details are hidden from third parties while still allowing the recipient to prove ownership and spend the coin.

Private Transactions. Flash supports two types of private transactions, each of which enables secure, anonymous transfer of value:

– **Mint Transactions:** These transactions create new coins and assign them to a recipient's address. Mint transactions specify a public value (the total amount) for the coin but keep the details of the recipient's identity and the coin's value hidden. Mint transactions include cryptographic proofs, which allow the recipient to verify the accuracy and authenticity of the transaction without revealing any private information.
– **Spend Transactions:** Spend transactions consume previously minted coins and generate new coins with equivalent secret value. These transactions allow the transfer of funds while maintaining the confidentiality of the transaction details. The spend transaction process ensures that the balance of the user is correctly maintained by using cryptographic proofs to validate that the spent coins were legitimate and properly issued.

Both transaction types make use of zero-knowledge proofs to ensure the correctness of the transaction without revealing sensitive information such as the amount or the identity of the sender and recipient.

Tags. Tags are an essential feature in preventing double-spending attacks. In the Flash protocol, each coin is associated with a unique identifier, referred to as a tag, which prevents the same coin from being spent more than once. These tags are publicly visible, allowing anyone to verify whether a coin has been spent, but they cannot be used to link specific coins to individual users or addresses. This ensures that the system can detect double-spending attempts without compromising user privacy. The full view key is required to correlate tags with specific coins, which means that only the owner of the view key can link the tags to the transactions.

Algorithms. The following algorithms are used to handle the generation, validation, and transfer of coins in the Flash system:

- Setup: This algorithm generates the public parameters of the system without relying on any trusted setup. These public parameters are used to define the cryptographic environment in which the system operates, ensuring that the protocol can function securely and verifiably in a decentralized manner.
- CreateKeys: This algorithm is used to generate the key pairs necessary for address creation, coin handling, and spending. It produces the incoming, full, and spend keys for each user, ensuring that they have the necessary cryptographic tools to interact with the system.
- CreateAddress: This algorithm generates a public address based on the user's key pair. These addresses are used to receive coins and track incoming funds, and they can be diversified to ensure privacy.
- CreateCoin: This algorithm is used to generate new coins. It takes a recipient's address and an amount of value to be transferred, and creates a coin that can be sent to the recipient. The coin is cryptographically secure and ensures that the details of the transaction remain private.
- Mint: This algorithm creates a mint transaction, transferring value from one address to another. It generates cryptographic proofs to validate the transaction while keeping the recipient's identity and transaction details hidden from third parties.
- Identify: This algorithm is used to identify whether a coin belongs to a particular recipient's address. It ensures that the ownership of the coin can be verified without revealing the recipient's identity or the value of the coin.
- Recover: This algorithm recovers additional data necessary for spending or verifying a coin. If a coin is spent, this algorithm retrieves the necessary cryptographic evidence to ensure the transaction is valid.
- Spend: This algorithm is used to consume coins and generate new coins with secret values. It ensures that the transaction is valid by using cryptographic proofs to confirm the legitimacy of the spent coins.
- Verify: This algorithm is responsible for validating transactions. It checks the authenticity of a transaction, ensuring that no double-spending has occurred and that the cryptographic proofs provided by the sender are valid.

These algorithms work in concert to provide a secure, private, and decentralized environment for conducting transactions. By leveraging zero-knowledge proofs, efficient key management, and cryptographic commitments, the Flash protocol ensures that users can transact securely without exposing their sensitive data.

Further details and an in-depth security analysis are provided in Appendix C, where the security properties of the protocol are rigorously examined.

4 Algorithm Constructions

This section describes the key algorithms in the DAP scheme.

4.1 Setup

Generates public parameters for the protocol.

Inputs: Security parameter λ, decomposition parameters n, m, maximum value $v_{\max}$. **Outputs:** Public parameters pp.

1. Sample a prime-order group $\mathbb{G}$ with field $\mathbb{F}$, and random generators $F, G, H, U \in \mathbb{G}$.
2. Define hash functions $\mathcal{H}_k, \mathcal{H}_{Q_2}, \mathcal{H}_{\text{ser}}, \mathcal{H}_{\text{val}}, \mathcal{H}_{\text{bind}}$, and $\mathcal{H}_{\text{div}}$.
3. Compute public parameters for Pedersen commitments, range proofs, encryption schemes, and Chaum-Pedersen proving systems.
4. Output all generated parameters as pp.

4.2 CreateKeys

Generates key tuples.

Inputs: Security parameter λ, public parameters pp. **Outputs:** Key tuple $(\mathsf{addr}_{\text{in}}, \mathsf{addr}_{\text{full}}, \mathsf{addr}_{\text{sk}})$.

1. Sample random values $s_1, s_2, r \in \mathbb{F}$.
2. Define $\mathsf{addr}_{\text{in}} = (s_1, P_2)$, $\mathsf{addr}_{\text{full}} = (s_1, s_2, D, P_2)$, $\mathsf{addr}_{\text{sk}} = (s_1, s_2, r)$.
3. Output $(\mathsf{addr}_{\text{in}}, \mathsf{addr}_{\text{full}}, \mathsf{addr}_{\text{sk}})$.

4.3 CreateAddress

Generates diversified addresses.

Inputs: $\mathsf{addr}_{\text{in}}$, diversifier i. **Outputs:** Diversified address $\mathsf{addr}_{\text{pk}}$.

1. Compute $d = \mathsf{SymEncrypt}(\mathsf{SymKeyGen}(s_1), i)$, $Q_{1,i}$, and $Q_{2,i}$.
2. Define $\mathsf{addr}_{\text{pk}} = (d, Q_{1,i}, Q_{2,i})$.

4.4 CreateCoin

Creates a new coin for a given address.

Inputs: $\mathsf{addr}_{\mathrm{pk}}$, value v, memo m, type bit b. **Outputs:** Coin Coin, nonce k.

1. Parse $\mathsf{addr}_{\mathrm{pk}}$, sample nonce k, compute K, S, and C.
2. For $b = 0$, generate a range proof Π_{rp} and encrypted recipient data $\overline{r}$.
3. Output Coin and k.

4.5 Mint

Generates coins with public value.

Inputs: Set of output addresses $\{\mathsf{addr}_{\mathrm{pk},j}, v_j, m_j\}_{j=0}^{t-1}$. **Outputs:** Mint transaction $\mathrm{tx}_{\mathrm{mint}}$.

1. Create output coins $\mathsf{OutCoins}$, parse commitments $\{\overline{C}_j\}$.
2. Generate representation proof Π_{val}.
3. Output $\mathrm{tx}_{\mathrm{mint}}$.

4.6 Identify

Determines if a recipient controls a coin.

Inputs: $\mathsf{addr}_{\mathrm{in}}$, coin Coin. **Outputs:** Value v, memo m, diversifier i, nonce k.

1. Parse Coin, decrypt recipient data r, and verify commitments.
2. Compute diversifier i and output (v, m, i, k).

4.7 Spend

Consumes coins to create new coins.

Inputs: $\mathsf{addr}_{\mathrm{full}}$, $\mathsf{addr}_{\mathrm{sk}}$, input coins, output addresses, fee f. **Outputs:** Spend transaction $\mathrm{tx}_{\mathrm{spend}}$.

1. For each input coin, generate serial and value commitment offsets and a one-out-of-many proof.
2. Create output coins $\mathsf{OutCoins}$ and generate balance proof Π_{bal}.
3. Generate a Chaum-Pedersen proof Π_{chaum} and output $\mathrm{tx}_{\mathrm{spend}}$.

5 Multisignature Operations

Multisignature operations enable transactions requiring authorization from multiple parties, without relying on a trusted third party. We employ techniques inspired by MuSig [14] and FROST [13], supporting threshold signing with efficient precomputation [8,**?**]. Security analysis is deferred to future work.

Given ν players and a threshold t, we outline collaborative methods for key generation ($\mathsf{CreateKeys}$), nonce precomputation ($\mathsf{Precompute}$), and transaction signing (Spend).

5.1 CreateKeys

1. **Key Setup**: Each player generates:

- Random polynomial coefficients and shares $s_{1,\alpha}, s_{2,\alpha} \neq 0$.
- Commitments and proofs of coefficient knowledge.

These are exchanged and verified. 2. **Key Aggregation**: Players compute:

$$r_\alpha = \sum_{\beta=1}^{\nu} \widehat{r}_{\beta,\alpha}, \quad s_1 = \sum_{\beta=1}^{\nu} \mathcal{H}_{s_1}(\{s_{1,\gamma}\}, \beta) s_{1,\beta}, \quad s_2 = \sum_{\beta=1}^{\nu} \mathcal{H}_{s_2}(\{s_{2,\gamma}\}, \beta) s_{2,\beta}.$$

All players verify completion before using keys.

5.2 Precompute

Players precompute π nonce pairs $(d_{\alpha,k}, e_{\alpha,k})$, share commitments, and verify correctness. These nonces are stored for future signing.

5.3 Spend

Threshold signing generates Chaum-Pedersen proofs: 1. Players use precomputed nonces to create commitments $A_1, \{A_{2,u}\}$ and challenge c. 2. Each player computes response shares $t_{2,\alpha}$ using Lagrange coefficients and exchanges them. 3. Aggregated responses produce:

$$\{t_{1,u}\} = \{\mathcal{H}_F(\rho_{u,\beta}) + c^u s_u\}, \quad t_2 = \sum_{\beta=1}^{t} t_{2,\beta}, \quad t_3 = \sum_{u=0}^{w-1} (\mathcal{H}_H(\rho_{u,\beta}) - c^u \mathcal{H}_{\text{ser}'}(s_u, D)).$$

This process ensures collaborative signing with minimal communication overhead.

6 View Keys and Installment Proofs

Flash empowers secure and adaptable exchange taking care of through its key designs: - **Approaching Perspective Key**: Utilized in Identify to recognize coins shipped off a location, their worth, and update information. This backings: 1. **Delegated Scanning**: Empowers filtering without spend authority. 2. **Secure Key Storage**: Keeps spend keys scrambled during checking. - **Full View Key**: Utilized in Recover for exchange discovery, total calculation, and oversight. It upholds: 1. **Public Oversight or Auditing**: Permits outsiders (e.g., examiners, good cause) to screen exchanges. 2. **Multisignature Monitoring**: Empowers cosigners to check spending activities.

The full view key additionally offloads computationally escalated evidences in Spend to strong gadgets, safeguarding enjoy expert on the gadget with the spend key.

Installment Proofs (Supplement **??**) uncover individual coin information in zero-information, demonstrating spend authority without uncovering more extensive exchange subtleties. Use cases include: - Demonstrating installments to retailers. - Specific exposure for gifts or reviews.

7 Efficiency

Exchange productivity is assessed by size, age, and confirmation intricacy. Key presumptions: - Coin values/expenses: 8 bytes, notices: M bytes, diversifiers: I bytes. - Confirmed encryption utilizes 16-byte labels and 32-byte responsibilities [1,**?**]. - Verifications are size-enhanced with shortened hashes [15].

Confirmation depends on straight mix assessments in $\mathbb{G}$, empowering cluster check through irregular weighting [17]. For B exchanges spending w coins and producing t coins, Table 2 subtleties the bunch intricacy as far as unmistakable $\mathbb{G}$ components.

Table 1. Spend transaction size by component

Component	Instantiation	Size ($\mathbb{G}$)	Size ($\mathbb{F}$)	Size (bytes)
f				8
Π_{rp}	Bulletproofs+	$2\lceil \lg(64t) \rceil + 3$	3	
Π_{bal}	Schnorr (short)		1.5	
Π_{chaum}	this paper	$w + 1$	$w + 2$	
Input data (w coins)				
(S', C')		$2w$		
Π_{par}	this paper	$(2m + 2)w$	$[m(n-1) + 3]w$	
Output data (t coins)				
(S, K, C)		$3t$		
$\overline{r}$	ChaCha20-Poly1305			$(8 + M + I + 48)t$

Table 2. Spend transaction batch verification complexity for B transactions with w spent coins and t generated coins

Component	Complexity
Parallel one-out-of-many	$B[w(2m + 2) + 2n^m] + 2mn + 1$
Bulletproofs+	$B(t + 2\lg(64t) + 3) + 128T + 2$
Modified Chaum-Pedersen	$B(3w + 1) + 4$
Schnorr	$B(w + t + 1) + 2$

The equal one-out-of-many demonstrating framework can be upgraded by pre-consolidating relating S_i and V_i components with a weight, lessening confirmation time, as displayed in starting tests.

References

1. Albertini, A., Duong, T., Gueron, S., Kölbl, S., Luykx, A., Schmieg, S.: How to abuse and fix authenticated encryption without key commitment. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 3291–3308. USENIX

Association, Boston, MA (Aug 2022), https://www.usenix.org/conference/usenixsecurity22/presentation/albertini
2. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) Advances in Cryptology — ASIACRYPT 2001. pp. 566–582. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
3. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
4. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: Proceedings of the 23rd USENIX Conference on Security Symposium. p. 781–796. SEC'14, USENIX Association, USA (2014)
5. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050 (2017), https://ia.cr/2017/1050
6. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334 (2018). https://doi.org/10.1109/SP.2018.00020
7. Chung, H., Han, K., Ju, C., Kim, M., Seo, J.H.: Bulletproofs+: Shorter proofs for privacy-enhanced distributed ledger. Cryptology ePrint Archive, Report 2020/735 (2020), https://ia.cr/2020/735
8. Crites, E., Komlo, C., Maller, M.: How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375 (2021), https://ia.cr/2021/1375
9. Goodell, B., Noether, S., RandomRun: Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Report 2019/654 (2019), https://ia.cr/2019/654
10. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. pp. 253–280. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
11. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification (2021), https://github.com/zcash/zips/blob/master/protocol/protocol.pdf
12. Jivanyan, A.: Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Report 2019/373 (2019), https://ia.cr/2019/373
13. Komlo, C., Goldberg, I.: FROST: Flexible round-optimized Schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852 (2020), https://ia.cr/2020/852
14. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to Bitcoin. Designs, Codes and Cryptography **87**(9), 2139–2164 (2019)
15. Neven, G., Smart, N.P., Warinschi, B.: Hash function requirements for Schnorr signatures. Journal of Mathematical Cryptology **3**(1), 69–87 (2009). https://doi.org/10.1515/JMC.2009.004
16. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
17. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)

18. Pyrros Chaidos, V.G.: Lelantus-CLA. Cryptology ePrint Archive, Report 2021/1036 (2021), https://ia.cr/2021/1036

A Modified Chaum-Pedersen Demonstrating System

The demonstrating framework is characterized by the connection:

$$\left\{pp_{\text{chaum}}, \{S_i, T_i\}_{i=0}^{l-1} \subset \mathbb{G}^2; (\{x_i, y_i, z_i\}_{i=0}^{l-1}) \subset \mathbb{F}^3 : \forall I, S_i = x_i F + y_i G + z_i H, U = x_i T_i + y_i G\right\}$$

The convention continues as follows:

1. Prover processes $A_1 = \sum_{i=0}^{l-1} r_i F + \sum_{i=0}^{l-1} s_i G + tH$ and sends $A_1, A_{2,i} = r_i T_i + s_i G$ to verifier.
2. Verifier challenges with $c \in \mathbb{F}$.
3. Prover sends reactions $t_{1,i}, t_2, t_3$.
4. Verifier checks:

$$A_1 + \sum_{i=0}^{l-1} c^{i+1} S_i = \sum_{i=0}^{l-1} t_{1,i} F + t_2 G + t_3 H \quad \text{and} \quad \sum_{i=0}^{l-1} (A_{2,i} + c^{i+1} U) = \sum_{i=0}^{l-1} t_{1,i} T_i + t_2 G.$$

Proof. The convention is finished. Exceptional adequacy follows from extricating values x_i, y_i, z_i through tackling straight frameworks. The convention is exceptional legitimate verifier zero-information by mimicking arbitrary difficulties and reactions.

B Parallel One-out-of-Many Demonstrating System

The framework demonstrates that S_l and V_l have a place with a set utilizing network responsibilities:

$$\{pp_{\text{oneoutofmany}}, A, B \in \mathbb{G}^n, S_i = \{S_{i,1}, \ldots, S_{i,n}\}, V \in \mathbb{G}, T \in \mathbb{G}, f \in \mathbb{F}\}$$

The convention continues as:

1. Prover processes responsibilities $A = \sum_{i=1}^{n} x_i S_i$ and sends them.
2. Verifier challenges with x.
3. Prover sends reactions and verifier actually takes a look at consistency.

The framework ensures trustworthiness and protection while forestalling fashioned evidences.

C Payment Framework Security

Zerocash [3] laid out a security model for decentralized mysterious installment (DAP) frameworks, where enemies can present vindictive coins, control exchange

inputs, and create erratic exchanges. We demonstrate Flash's security in a comparable model, where DAP is a tuple of calculations:

(Setup, CreateKeys, CreateAddress, CreateCoin, Mint, Identify, Recover, Spend, Verify)

The framework fulfills culmination, balance, non-pliability, and record vagary in the event that no foe can break the properties.

The prophet $\mathcal{O}^{\mathsf{DAP}}$ reenacts the way of behaving of legit parties with a connection point for executing the key capabilities, including CreateAddress, Mint, and Spend. The foe collaborates with the prophet through questions, which incorporate making addresses, stamping coins, and spending them.

C.1 Completeness

Culmination guarantees that no foe can keep a legit client from spending a coin. In the event that a client can recognize a coin with the approaching perspective key, it can recuperate and spend it utilizing the full view key. This holds since, supposing that an enemy controls a coin, it should be reflected in a past substantial exchange, and a fair client can't create a spend exchange without the right key.

C.2 Balance

Balance guarantees that an enemy have zero control over additional coins than are printed or spent to it. The foe wins in the event that it controls more worth than lawfully permitted. In the BAL game, the foe's unspent coins in addition to coins spent to it shouldn't surpass the stamped or got coins.

Lemma 1. *Given a record, two in any case substantial spend exchanges uncover similar label provided that the coins' sequential responsibilities are separated in the structure:*

$$S_1 = \mathsf{Comm}(x, y, \beta_1), \quad S_2 = \mathsf{Comm}(x, y, \beta_2)$$

Proof. For two exchanges with label T, every exchange's legitimate evidence yields articulations and witness esteems that should match for the exchanges to be substantial, prompting extricated sequential responsibilities S_1 and S_2 with the equivalent x and y values, contrasting just in β_1 and β_2.

This outcome likewise applies to copy labels in a similar exchange.

C.3 Transaction Non-Malleability

Exchange non-pliability guarantees that no enemy can change a substantial exchange. For spend exchanges, an enemy $\mathcal{A}$ connects with a prophet $\mathcal{O}^{\mathsf{DAP}}$ and yields an exchange tx'. On the off chance that tx is a substantial exchange from $\mathcal{O}^{\mathsf{DAP}}$, $\mathcal{A}$ wins if:

- $\mathrm{tx}' \neq \mathrm{tx}$,
- tx' shares a tag with tx, and
- both tx' and tx are substantial comparative with a similar record prefix.

A DAP conspire $\mathit{\Pi}$ is TRNM-secure if:

$$\Pr[\mathsf{TRNM}(\mathit{\Pi}, \mathcal{A}, \lambda) = 1] \leq \mathrm{negl}(\lambda).$$

C.4 Ledger Indistinguishability

Record vagary guarantees that no enemy $\mathcal{A}$ can recognize communications with two records L_b and L_{1-b} created by prophets $\mathcal{O}_b^{\mathsf{DAP}}$ and $\mathcal{O}_{1-b}^{\mathsf{DAP}}$. $\mathcal{A}$ inquiries these records and surmises a piece b'. A DAP conspire $\mathit{\Pi}$ is LIND-secure if:

$$\Pr[\mathsf{LIND}(\mathit{\Pi}, \mathcal{A}, \lambda) = 1] - \frac{1}{2} \leq \mathrm{negl}(\lambda).$$

C.5 Payment Proofs

Installment confirmations guarantee:

- Confirmations can't be replayed.
- Provers know the coin's mystery key.
- Verifiers affirm the coin's worth and notice.
- Beneficiaries recognize coins utilizing view keys.
- Foes can't fashion verifications for various addresses.

Protocol: The prover ties coin information, setting, and keys into a Chaum-Pedersen confirmation $\mathit{\Pi}_{\mathrm{auth}}$, remembered for the installment verification $\mathit{\Pi}_{\mathrm{pay}} = (\mathrm{Coin}, k, d, Q_1, Q_2, \mathit{\Pi}_{\mathrm{auth}})$. Check affirms confirmation legitimacy, unscrambling consistency, and restricting accuracy.

Security: Setting restricting forestalls replay assaults. Decoding approves coin information, and verification imperatives guarantee foes can't create substantial confirmations for various addresses.