Cryptocurrency-Backed Trustless Anonymous Tokens and Their Applications

Amit Agarwal*, Kushal Babel*, Sourav Das*, Ari Juels†‡, Peter Rindal*, Aayush Yadav§
*Category Labs
†Cornell Tech
‡IC3
§George Mason University

***Abstract*—Public blockchains like Ethereum deliver transparency, but adding anonymity remains a fundamental challenge. Existing proposals either offer limited anonymity guarantees or rely on heavy cryptographic machinery, e.g., zero-knowledge proofs.**

We introduce *Blockchain Anonymous Tokens* (BAT), a system for efficient *sender*-anonymous transactions on transparent blockchains. Building on the observation that *one-time-spendable tokens suffice for many applications*, BAT has a lightweight design using classic anonymous tokens due to Chaum (1983). Unlike such tokens, though, BAT is designed to work in a transparent decentralized setting, where issuers are untrusted (i.e., any single potentially malicious entity can be the issuer) and spends happen publicly. BAT issuance is compact: A client can receive ℓ tokens with just $\mathcal{O}(1)$ on-chain communication and computation.

We formalize the notion for BAT, and provide a concretely efficient construction. Our BAT construction requires no on-chain verification of expensive zero-knowledge proofs; just a signature verification during spends and a single exponentiation on-chain during issuance. We present several applications of BAT in various blockchain contexts.

We prove the security of our BAT scheme in the Generic Group Model and random oracle model and with any secure digital signature scheme.

We implement and evaluate BAT and show that compared to the closest baseline, Zcash transactions, BAT tokens are 50× shorter, 9× faster to verify, and 7,000× faster to generate.

1. Introduction

Transparent blockchains such as Ethereum do not provide anonymity to users, only the weaker privacy afforded by pseudononymous user addresses [1]–[3]. This limitation has motivated the development of smart-contract-based anonymity tools, such as Tornado Cash [4] and Privacy Pools [5], that enhance user anonymity. These tools, however, have seen very limited use. This is due in part to their reliance on heavy-weight cryptographic techniques such as zero-knowledge proofs, which, in the resource-constrained blockchain environments, do not yet scale to high transaction throughputs. Consequently, these systems are only suitable for low-throughput applications, such as one-off private payments, but not for high-throughput environments or low-value usecases, such as fee payments. The need for private fee payment in particular is an acknowledged bottleneck in the construction of private transaction systems [6], [7].

In this paper, we present Blockchain Anonymous Tokens (BAT), a lightweight, efficient cryptographic system that provides anonymity to payers or transaction senders on transparent blockchains. Our key observation is that for many applications (such as fee payments), users only need *one-time-spendable* anonymous currencies, not general-purpose privacy coins like Zcash that also hide recipients and transaction amounts and enable multiple payment hops. BAT is concerned with providing only transaction sender anonymity; hiding other transaction data, such as recipient address, amount, or smart contract calldata, is out of scope. BAT's minimalist approach leads to a privacy-preserving payment system with superior performance (in both computation and communication) to general-purpose schemes, yet with a range of important practical applications.

A prime example of such an application is the pervasive problem of front-running and Maximal Extractable Value (MEV) [8]–[11], where miners or validators can exploit knowledge of pending transactions in the mempool to censor, reorder, or insert their own transactions for profit. While solutions such as encrypted mempools [12]–[19] aim to hide transaction data to mitigate MEV attacks, they do not address the problem of sender metadata. This is because sender metadata (specifically, sender address) is required at the time of transaction inclusion in many blockchain designs in order to verify payment ability and thus prevent DoS attacks. But such metadata can itself facilitate MEV attacks—despite encryption of transaction data such as recipient and smart contract calldata [7]. BAT is particularly well-suited as a privacy-preserving alternative, as fee payments must happen with high throughput, low latency, and small communication overhead. Fee payments must also (in most cases) be denominated in the underlying currency of the blockchain, a property BAT achieves.

We instantiate BAT as a form of classic e-cash system [20], but one that is backed by cryptocurrency and operates on top of a smart-contract enabled blockchain platform. This allows us to effectively solve a key problem of requiring a trusted issuer (or "bank") in the e-cash and related anonymous tokens literature. Unlike e-cash, BAT

tokens can be issued by any *untrusted* entity with sufficient cryptocurrency stake in the blockchain. Stated differently, BAT effectively allows anyone to start up their own "bank". While e-cash like systems have been used previously for blockchain privacy, e.g., in Chaumian Coinjoin and other mixers, those systems are often tied to the mechanics of Bitcoin and designed for synchronous privacy operations over participating users. BAT can operate asynchronously in any smart-contract enabled setting.

Finally, BAT offers flexible deployment: it can operate natively in a blockchain—scaling to the blockchain's full transaction load—or can be deployed as a smart contract to provide anonymity at the application layer.

Challenges. Blockchains present unique challenges for designing anonymous token schemes such as BAT:

- *On-chain efficiency.* Blockchains are a highly resource-constrained environment. Therefore, the on-chain operations for issuance of BAT tokens and spend must be highly efficient in both communication and computation.
- *Permissionless issuance.* BAT is meant to work with *untrusted* issuers without involving trusted committees running expensive threshold cryptography. To achieve this, we need to overcome two challenges: (i) a malicious issuer should not be able to cause clients to pay but refuse to issue tokens, or issue unspendable tokens and (ii) a malicious issuer should not be able to issue more tokens (e.g., off-chain in a black market) than the payments made by clients on-chain for issuance; this would lead to insolvency, i.e., more tokens redeemable than paid for.
- *Public spends.* Due to the public nature of pending blockchain transactions and the anonymous nature of the tokens, BAT must ensure that one client cannot steal tokens from another client despite the tokens being broadcast publicly to the blockchain. Furthermore, for various applications such as paying fees anonymously for blockchain transactions, the tokens should enable binding to the client's chosen associated data after issuance.

Contributions. We make the following contributions:

- We formalize correctness and security definitions for cryptocurrency-backed anonymous token schemes, including unlinkability (sender anonymity), unforgeability, issuer fairness (honest issuer against malicious clients), client fairness (honest clients against malicious issuers), and a novel solvency condition that captures the 1:1 backing of tokens by the underlying cryptocurrency.
- We design a BAT scheme with constant on-chain overhead (only one group exponentiation) for issuing tokens in arbitrary batches and an efficient spend procedure that involves only two signature verifications.
- We prove security of our BAT scheme in the Generic Group Model (GGM), the Random Oracle Model (ROM), and any secure digital signature scheme.
- We implement and benchmark the performance of all components, including client-side, issuer-side, and on-chain costs. Using recent benchmarks for Zcash Orchard as a baseline [21], we show that BAT tokens are more than 50× smaller in size, about 9× faster to verify, and over 7,000× faster to generate than Zcash proofs.

Limitations. One limitation of our BAT scheme, compared to systems such as Zcash, is that it provides issuer-specific anonymity rather than global anonymity. In particular, for any given issuer, the anonymity set of a token is limited to the set of clients that obtained tokens from that issuer. Moreover, as in Zcash, we also need to maintain an ever-growing set of spent tokens ("the nullifier set") for each issuer. However, in BAT, we can manage this set by periodically retiring issuers; once we retire an issuer, we can safely forget the state associated with that issuer.

Paper organization. We present a technical overview of our construction in Section 2. In Section 3, we discuss some applications of BAT and related work, and present formal definitions and security notions for BAT in Section 4. We describe our construction in Section 5, analyze it in Section 6, and provide an evaluation of our implementation in Section 7. Finally, we discuss various deployment considerations for BAT in Section 8.

2. Technical Overview

Notation. We use λ to denote a computational security parameter, $[a, b]$ to represent the set of integers $\{a, a+1, \ldots, b\}$ and $[n] \coloneqq [1, n]$, $x \leftarrow\!\!\$\ S$ to denote that x is an element sampled uniformly at random from set S. For a vector $\boldsymbol{v}$ of length n, we use the notation v_i to indicate the i^{th} element of $\boldsymbol{v}$ where $i \in [n]$. By $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$, we mean the class $\lambda^{O(1)}$ and $\frac{1}{\lambda^{\omega(1)}}$. For security parameter λ, we use PPT to denote probabilistic $\mathsf{poly}(\lambda)$-time Turing Machines with $\mathsf{poly}(\lambda)$-sized advice.

Bilinear groups. A bilinear group is a set of three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T})$ of prime order p, with a (non-degenerate) bilinear map or pairing $\circ : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$. The groups $(\mathbb{G}_1, \mathbb{G}_2)$ are referred to as the source groups, while $\mathbb{G}_\mathsf{T}$ is the target group. The groups $\mathbb{G}_1, \mathbb{G}_2$ have generators $[\![1]\!]_1, [\![1]\!]_2$. Let GGen be the group generation algorithm, i.e., $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, \circ) \leftarrow \mathsf{GGen}(1^\lambda)$.

Figure 1 shows an overview of our protocol.

A BAT protocol involves three parties: clients who purchase anonymous tokens, issuers who issue these tokens, and a blockchain environment $\mathcal{E}$ that enforces the atomicity of issuance, and validates anonymous token spends and refunds for unspent tokens. Each token represents a fixed denomination of one unit of cryptocurrency.

Desired properties. As we discuss in Section 1, informally, a BAT protocol must satisfy the following security properties. (i) *Client anonymity:* Spent tokens can not be linked to its issuance. (ii) *Issuer-fairness:* Malicious clients cannot spend or refund tokens they did not legitimately obtain. (iii) *Client-fairness* A client who pays for a token can either publicly spend the token successfully or get a refund for it. (iv) *Solvency:* A malicious issuer can not mint anonymous tokens without paying in the corresponding amount of cryptocurrency on-chain.

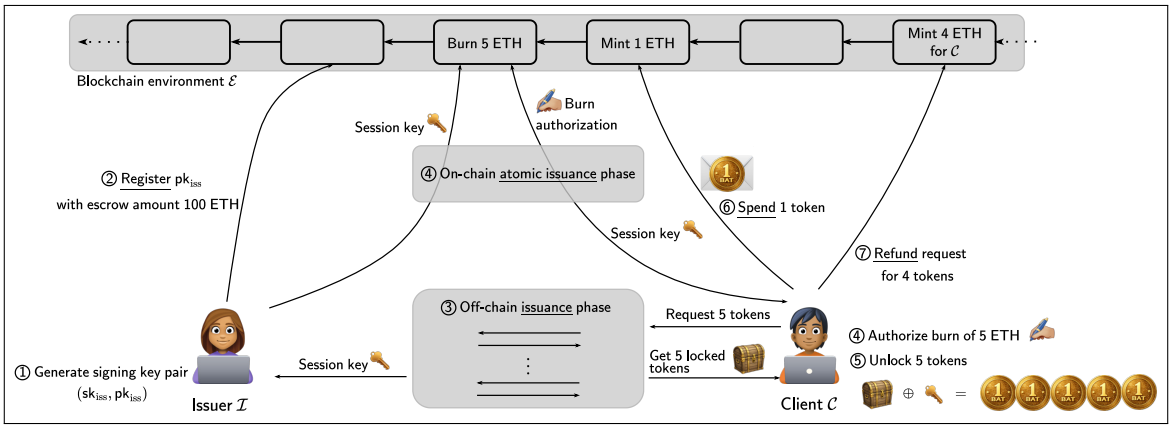


Figure 1: *Overview of our blockchain anonymous token (*BAT*) architecture involving three entities - client, issuer, blockchain environment - and four main phases including registration, issuance, spend and refund. The circled numbers denote the sequence in which various steps are executed. The grey box depicting the issuance phase indicates that the phase is an interactive protocol. Here we abstract away the details of various checks that are performed by different entities for the ease of illustration.*

In addition to security, the protocol must achieve practical efficiency, particularly in on-chain costs and the issuer's computational and communication overhead.

We first describe two baseline protocols, and their limitations to build intuition for our protocol.

Baseline 1: Protocol without compact issuance. Consider a protocol in which, for each anonymous token, the client interacts with the issuer to obtain a blind signature. The client then converts the blind signature into an unlinkable anonymous token using standard techniques and spends the token on-chain. To enforce fairness, the client and the issuer route all interactions of the blind signature protocol through the blockchain environment $\mathcal{E}$. Concretely, the client first escrows the payment for the token on-chain. The blockchain releases this payment to the issuer only if the on-chain transcript of the blind signature protocol satisfies a pre-specified predicate. This predicate ensures that, given a valid transcript, the client can derive a valid anonymous token.

As described, this protocol does not provide solvency: a malicious issuer can locally mint unbounded number of tokens and spend them on-chain without paying for them. Moreover, the protocol incurs prohibitively high on-chain costs. To issue ℓ tokens, the client and issuer must post $O(\ell)$ communication on-chain, and $\mathcal{E}$ validates them all.

Baseline 2: BAT **using fair-data exchange.** One approach for lowering the on-chain communication and computation cost is to use the fair-data exchange protocol from [22]. Briefly, using a fair-data exchange protocol, the issuer encrypts its blind signature messages and sends the ciphertexts to the client off-chain, together with a zero-knowledge proof that the ciphertexts encrypt valid blind signature messages. The client and issuer then execute a fair-exchange protocol on-chain to exchange payment for the decryption key. Although this approach reduces on-chain communication and computation, it significantly increases the issuer's workload. The issuer must generate expensive zero-knowledge proofs, and it must do so for many clients concurrently. This requirement creates a substantial computational bottleneck and limits the practicality of the approach. We note that as in the first baseline, this protocol also does not guarantee solvency.

Overview of our protocol. Unlike the baselines, our protocol achieves both on-chain and off-chain efficiency: on-chain overhead is constant and independent of the number of tokens issued in a batch, and our protocol only adds a marginal overhead at the issuer compared to the standard signature generation. Similarly, the additional off-chain cost at the client involves one group exponentiation per token.

Looking ahead, we adopt the ideas in the second baseline protocol, which achieves constant on-chain overhead using the fact that the blockchain does not need to verify each individual blind signature during issuance. Instead, the blockchain only needs to ensure that the issuer has provided a valid decryption key. To lower the issuer's overhead in this baseline, we instantiate both the blind signature scheme and the fair-data exchange protocol in a non-black box manner. In particular, in our scheme, we use the classic BLS blind signature scheme and rely on the underlying algebraic structure to efficiently mask ℓ blind signatures. Next, we provide more details of our protocol and discuss how it ensures solvancy in presence of a malicious issuer.

Our protocol has five logical phases: *registration*, *issuance*, *spend*, *refund*, and *retirement*. We describe each phase below assuming ETH as the underlying currency.

Registration phase. In this phase, any party that holds a private-public key pair $(\mathsf{sk}_{\mathsf{iss}}, \mathsf{pk}_{\mathsf{iss}}) \in \mathbb{Z}_p \times \mathbb{G}_2$ registers as an issuer by escrowing ℓ_{iss} units of ETH on the blockchain.

Issuance phase. In this phase, a client requests ℓ tokens from an issuer off-chain. Each token is a blind signature on an ephemeral public key of a digital signature scheme. Concretely, we use the BLS blind signature [23] where for each $i \in [\ell]$, the i-th blind signature is defined as $\tilde{\sigma}_i :=$

$\mathsf{sk}_{\text{iss}} \cdot \mathsf{H}(M_i) \in \mathbb{G}_1$, for a client-chosen message M_i and a hash function H mapping messages into $\mathbb{G}_1$.

To ensure that the client pays for the requested tokens, the issuer does not send the blind signatures $\{\tilde{\sigma}_i\}_{i\in[\ell]}$ directly. Instead, the issuer sends masked blind signatures. Specifically, the issuer samples a session key $k \in \mathbb{Z}_p$ and sends the masked signatures $\{k \cdot \tilde{\sigma}_i\}_{i\in[\ell]}$ to the client. Here, using the same session key k for all ℓ tokens is critical for on-chain efficiency. Additionally, as a proof, the issuer sends a commitment of k which we define as $\mathsf{com}_k := k \cdot \mathsf{pk}_{\text{iss}}$. As we show in Section 5, the client can use com_k to verify the correctness of the *masked* signatures with no additional overhead beyond standard BLS blind-signature verification. Our protocol adds marginal overhead at the issuer.

Finally, to close the issuance session, the client and the issuer atomically exchange the session key k and the corresponding payment on-chain. During this exchange, the blockchain performs the following checks: (i) the number of tokens issued by pk_{iss} so far is less than $\ell_{\text{iss}}/2$, i.e., half of pk_{iss}'s escrowed collateral; and (ii) $k \cdot \mathsf{pk}_i = \mathsf{com}_k$. The first check enforces solvency, while the second ensures that the client can recover k and thus obtain the blind signatures. Both checks incur minimal on-chain cost.

Spend phase. Let $(\mathsf{sk}_{\text{eph}}, \mathsf{pk}_{\text{eph}})$ be an ephemeral secret-public key pair such that: (i) the client knows sk_{eph}, and (ii) the client obtained a (blind) signature σ on pk_{eph} during the issuance phase. Given the tuple $(\sigma, \mathsf{sk}_{\text{eph}}, \mathsf{pk}_{\text{eph}})$ and associated data $\mathsf{ad} \in \{0,1\}^*$ (e.g., a transaction), the client generates an ephemeral signature γ on ad using sk_{eph}. The client posts $\rho := (\sigma, \mathsf{pk}_{\text{eph}}, \gamma, \mathsf{ad})$ as an anonymous token.

Upon receiving ρ, the blockchain performs the following checks: (i) σ is a valid BLS signature on pk_{eph}, confirming that the claimed issuer issued the token; (ii) γ is a valid signature on ad under pk_{eph}, ensuring the owner's spend authorization is bound to ad; (iii) pk_{eph} does not appear in the set of previously spent tokens, preventing double-spends; and (iv) the total number of token spends for pk_{iss} does not exceed the number of tokens issued by pk_{iss}, which as we discuss in Lemma 6.3 is critical to ensure solvency.

If all checks succeed, the blockchain accepts the token, and marks pk_{eph} as spent. Additionally, it increments the spend counter associated with pk_{iss}, and mints one token-equivalent worth of cryptocurrency for consumption.

Refund phase. In our protocol, a client can obtain refunds for ℓ_r unspent tokens by submitting them to the blockchain. Upon receiving a refund request, the blockchain verifies that (i) the tokens correspond to a valid on-chain issuance, (ii) the tokens are valid, and (iii) the tokens are unspent. If all checks pass, the blockchain marks these tokens as spent, mints ℓ_r units of cryptocurrency, and credits them to the public key specified by the client during issuance. We note that the refund protocol must ensure that an adversary cannot use a refund request to frontrun the client and spend the tokens before the client can claim the refund.

Our basic refund protocol requires the client to submit a refund request and incurs $\mathcal{O}(\ell_r)$ on-chain communication and computation. In Section 5, we present an optimized protocol that reduces on-chain communication to $\mathcal{O}(1)$ and enables *automated* refunds without an explicit client request.

Retirement phase. In this phase, a registered issuer submits an on-chain transaction declaring its intent to retire. Upon receiving this request, the blockchain checks whether the total number of spends and refunds associated with the issuer exceeds the number of tokens the issuer issued. If it does, the blockchain recovers the excess amount from the issuer's collateral, possibly applying penalty. As we argue in the proof of Lemma 6.3, this suffices to ensure solvency.

3. Applications and Related Work

3.1. Applications

To motivate BAT, with its lightweight design choices—specifically, one-time-spendable tokens and sender-only anonymity—we briefly describe three example applications.

MEV mitigation. As touched on in Section 1, Maximal Extractable Value (MEV) [8]–[11] remains a pervasive challenge in blockchains. MEV refers to the practice of strategically manipulating transaction ordering and transaction inclusion for profit.

MEV exploitation depends upon adversarial knowledge of transaction information prior to block finalization. A widely considered countermeasure against harmful forms of MEV is encrypted mempools [24]–[27]. Encrypted mempools, though, require that metadata—specifically sender payment data—be revealed, so as to ensure fee payment. Otherwise, an adversary can mount costless denial-of-service attacks. Sender metadata, however, opens up an encrypted mempool to MEV, i.e., frontrunning of target senders' transactions.

Sender history alone can defeat the protection that encrypted-mempool calldata is supposed to provide [7]. Consider an Ethereum user account 0x8001…3fd1 that, across its 107 transactions, submits 104 WETH→BANANA swaps of roughly uniform size. An MEV bot who sees only the sender field of this account's next encrypted transaction can predict the swap with high confidence and frontrun it—before the calldata is ever revealed.

BAT can address this problem by removing the need for transactions to include sender payment data. Instead, senders can pay fees with BAT tokens. One-time token use suffices, and as the receiver is a blockchain's transaction-ordering system, receiver anonymity isn't needed.

Coercion-resistant on-chain voting. On-chain voting conducted by governance systems such as DAOs are susceptible to bribery and other forms of *coercion* if voter identities are known [28]. Bribery is a practical issue for DAOs [29].

Coercion-resistant voting systems, e.g., [30], [31], require use of encrypted ballots. As in the case of MEV, however, preventing DoS attacks requires voters to pay fees. BAT enables voters to pay fees anonymously, thereby addressing DoS attacks in these systems. The voting system of course need not receive payments anonymously.

Private-payment on-ramps and solving the "relayer problem". Smart-contract systems such as Zether [32] enable private payments on transparent blockchains. However, they require users to fund private accounts with public accounts, thereby revealing the identity of the funding account. BAT can remove this linkage: a user first exchanges cryptocurrency for BAT tokens and then uses these tokens to fund private accounts anonymously. Token transferability isn't required for the on-ramp in these systems, as the private-payment system itself enables subsequent payments.

A similar problem has plagued privacy protocols on Layer 1 blockchains (Ethereum) such as Tornado Cash [4], Railgun [33], and Privacy Pools [5]: the user must pay gas to submit the deposit or withdrawal transaction, but paying from any wallet defeats the very anonymity provided by the pool. Deployed systems address this with *relayer networks*—third-party operators who submit a user's (withdrawal) transaction and pay gas on her behalf, and recover the gas cost (plus a fee) from the withdrawn funds [4], [5], [33]. Relayer networks reintroduce a trusted third party for transaction submission [34], and constitute a centralization, liveness, and regulatory pressure point in deployed privacy systems while degrading user experience. BAT obviates the relayer dependency entirely. A user obtains anonymous tokens from an untrusted issuer and then attaches them directly to her transactions to pay gas for interacting with privacy pools. Note that the underlying blockchain would need to support BAT for this application.

3.2. Related Work

Anonymous tokens and blockchain applications. Anonymous tokens allow a user to obtain and later redeem a token while preserving privacy which ensures that the issuer cannot link redemption to issuance. These tokens let users prove possession of authorization without revealing their identity or linkability. This unlinkability is essential in applications such as private web browsing (e.g., Privacy Pass) [35], [36], private contact tracing [37], fraud detection [37], and private click measurement [38]. These applications have inspired a rich body of research on anonymous tokens and blind signatures, with different trust, efficiency, and security tradeoffs [35], [37], [39]–[43].

Anonymous tokens were originally devised as the basis for anonymous electronic cash (e-cash) systems [44], [45]. In this setting, a user deposits funds with a trusted bank and receives an anonymous token that the user can later transfer or redeem without revealing their identity, even to the bank.

Coin mixing and tumbling services. Cryptocurrency mixers for Bitcoin, such as CoinJoin [46], CoinShuffle [47], [48] and TumbleBit [49], [50] aim to provide transaction unlinkability by shuffling funds among multiple participants. Tornado Cash [4], Möbius [51] and MixEth [52] provide a similar service on Ethereum. In these systems, the users deposit assets into a smart contract and later withdraw them using a cryptographic proof of deposit, thereby severing the on-chain link between source and destination addresses. However, these services either rely on costly multi-party coordination [47], expensive cryptographic primitives such as zk-SNARKs in Tornado Cash [4] or ring signatures in Möbius [51], which often limits anonymity sets; or incur high latency due to epoch-based designs [49], [50]. MixEth [52] instead mixes by verifiably shuffling the deposited public keys: any party may shuffle, but each shuffle is followed by a challenge period during which receivers must come online, verify the shuffle, and submit a fraud proof if their key was corrupted; funds can be safely withdrawn only after the desired number of shuffle rounds and their challenge periods have elapsed, and the anonymity set is bounded by the keys deposited into the shuffle pool. By contrast, BAT allows users to act independently and achieves anonymity via blind signatures, with on-chain costs limited to efficient signature verification, resulting in substantially lower overhead. For instance, in MixEth, a user's shuffle-and-deposit transaction alone consumes $366{,}216 + 10{,}000\,n$ gas for an n-key pool plus a 113,265-gas withdrawal—about 1.5M gas at $n = 100$ and growing with the anonymity set—whereas issuing and spending a BAT token costs ≈200k gas end-to-end, independent of the anonymity-set size (Section 7.3).

The use of e-cash-inspired anonymous tokens for mixing has been explored in CoinJoin [46] and in the 'blindly signed contracts' of Heilman, et al. [49]. The overall idea is familiar: a user escrows funds with an untrusted intermediary and engages in a fair-exchange protocol to obtain an anonymous token. The token can then be transferred to a recipient, who redeems it to claim escrowed funds, thereby breaking the link between sender and receiver on the blockchain.

These protocols have several limitations. Most notably, they are tailored to pairwise payments in UTXO-based systems and do not generalize to broader blockchain applications. Further, while the CoinJoin protocol is vulnerable to DoS attacks due to unmitigated aborts by mix participants [53], later improvements have had to rely on complex multi-party systems [47], [54]. In comparison, [49] avoids such coordination but instead requires participants to synchronize across epochs (due to timing attacks), which introduces latency and limits flexibility. Beyond these liveness and coordination issues, the lack of a refund mechanism enables 'griefing' attacks. For example, if Bob colludes with the intermediary and refuses to post the anonymous token, Alice has no mechanism to recover her escrowed funds.

BAT generalizes this paradigm, where tokens can be used across a wide range of applications beyond pairwise payments. Our design is epochless and avoids the timing attacks that motivated epoch-based constructions in prior work. We also provide the first formal treatment of this notion, including precise game-based security definitions and full security proofs. Moreover, BAT introduces an explicit refund mechanism that allows clients to recover funds associated with unspent tokens, a feature not addressed by existing approaches.

Blockchain-based confidential payment systems. Confidential payment systems aim to hide transaction flows

and associated metadata, such as account balances, on the blockchain. Early UTXO-based systems such as Zerocoin [55] and Zerocash (Zcash) [56] rely on zk-SNARKs to provide strong anonymity, but incur high computational costs or require a trusted setup. Monero [57] avoids trusted setup by using ring signatures [58], at the cost of achieving only k-anonymity and linear verification overhead in k.

These systems typically operate as separate ledgers and offer limited interoperability with existing smart-contract platforms. In the account-based model, Zether [32] provides a confidentiality layer for anonymous payments that can be implemented as a smart contract. However, their anonymity guarantee introduces significant overhead that scales linearly with the size of the anonymity set, and while followup works [59] have reduced this growth to logarithmic, our BAT achieves constant on-chain overhead for both issuance and spending, independent of the anonymity set size.

The UTT system of Tomescu et al. [60] provides more efficient confidential payments than Zcash. In principle, UTT can be used to implement a BAT protocol by removing support for multiple denominations, recipient hiding, and multi-hop payments. However, even with these simplifications, UTT has two critical limitations. First, it relies on threshold issuance, and, second, spending UTT tokens requires an expensive zero-knowledge proof, in addition to signature verification.

Anonymous payments with authorities, and on open chains. A complementary line of work builds private payments anchored to designated authorities: zkLedger [61] provides auditable, participant-hiding transfers over a permissioned ledger of banks, Platypus [62] and PEReDi [63] are central-bank digital currencies with privacy-preserving regulation, PARScoin [64] is a stablecoin with threshold issuance and auditing, and DART [65] carries issuer-designated auditing into the permissionless setting. In all of these, issuance, auditing or ledger maintenance rests with parties trusted (individually or as a threshold) to play their role; BAT's trust model is the opposite—a single issuer that may be arbitrarily malicious and is kept honest only by collateral on a permissionless chain. A second line achieves anonymous payments on open chains, but at a significant on-chain cost: QuisQuis [66] attaches zero-knowledge proofs over "updatable public keys", Veksel [67] attaches set-membership proofs for the spent coins, PriFHEte [68] achieves full privacy in the account model at the cost of heavy homomorphic computation by the miners, and OCash [69] extends strong anonymity to light clients by relying on ORAM techniques that have inherent overheads. A BAT spend, by contrast, verifies two signatures and requires no committee.

Stealth addresses. Stealth addresses [57], [70] let a sender derive a fresh one-time address per payment, hiding the *recipient* among all users. They complement BAT: a stealth address hides the payee, while BAT hides the payer.

4. Definition of Blockchain Anonymous Token

A *blockchain anonymous token* (BAT) protocol is a protocol between a client and an issuer involving a transparent payment environment $\mathcal{E}$ as a trusted third party (TTP) (or a blockchain). The TTP $\mathcal{E}$ manages accounts where each account is identified by a public key and has some non-negative money mapped to it. Throughout the paper, we use ETH to denote the denominations of the money. The TTP $\mathcal{E}$ can mint and burn ETH, or transfer ETH from one participant's account to another upon receiving an authorization from the sender in the form of a digital signature. Moreover, we assume that any messages sent to $\mathcal{E}$ are authorized and are received by $\mathcal{E}$ (i.e., included in the blockchain when we implement $\mathcal{E}$ using a blockchain) within a known bounded delay denoted by δ_{inc}. The environment $\mathcal{E}$ also maintains an increasing block-height counter $\mathcal{E}.\mathsf{currentHeight}$.

4.1. Interfaces

A blockchain anonymous token protocol BAT consists of the following algorithms (or protocols), where for any algorithm (or protocol) A, we use $\mathsf{A}^{\mathcal{E}}$ to indicate that the execution of A involves the TTP $\mathcal{E}$ and potentially triggers state changes in $\mathcal{E}$. We use the prefix Π to indicate an interactive protocol between the client and the issuer.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. The setup is a randomized algorithm that outputs the public parameters for the system (e.g., the description of appropriate spaces). All algorithms and protocols below implicitly take the pp as input.
- $\mathsf{IssKGen}(\mathsf{pp}) \to (\mathsf{sk}_{\mathsf{iss}}, \mathsf{pk}_{\mathsf{iss}})$. An issuer uses the $\mathsf{IssKGen}$ algorithm to sample a secret issuing key $\mathsf{sk}_{\mathsf{iss}}$ and the corresponding public verification key $\mathsf{pk}_{\mathsf{iss}}$.
- $\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathsf{iss}}, \ell_{\mathsf{iss}}) \to 0/1$. Any party uses the registration interface $\mathsf{Register}$ with its public key $\mathsf{pk}_{\mathsf{iss}}$ and deposit amount ℓ_{iss} to register itself as an issuer to $\mathcal{E}$. Upon successful registration, $\mathcal{E}$ transfers ℓ_{iss} ETH from the issuer to a freshly generated issuer specific escrow, and outputs 1. It outputs 0 otherwise.
- $\mathsf{Retire}^{\mathcal{E}}(\mathsf{pk}_{\mathsf{iss}}) \to 0/1$. Any existing issuer uses the retire interface with its public key $\mathsf{pk}_{\mathsf{iss}}$ to recover its remaining escrow. Upon successful execution, $\mathcal{E}$ returns $\ell'_{\mathsf{iss}} \in \mathbb{N}$ ETH from the issuer's escrow account to the issuer and outputs 1. Otherwise, it outputs 0.
- $\Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathsf{iss}}, \ell, \mathsf{sk}_{\mathsf{iss}}, \mathsf{st}_{\mathcal{I}}) \to (\mathsf{sid}, \mathsf{st}_{\mathcal{I}}, \mathsf{st}_{\mathcal{C}})$. The masked issuance protocol is a two-party protocol between a client and an issuer with public inputs being the issuer's public key $\mathsf{pk}_{\mathsf{iss}}$, the number of requested tokens ℓ, and the issuer's private input being its secret issuance key $\mathsf{sk}_{\mathsf{iss}}$, and its internal state $\mathsf{st}_{\mathcal{I}}$. The protocol outputs a public session identifier sid. Additionally, it outputs the modified issuer's internal state and privately outputs its internal state $\mathsf{st}_{\mathcal{C}}$ to the client.
- $\Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}, \ell, \mathsf{st}_{\mathcal{I}}, \mathsf{st}_{\mathcal{C}}) \to (\mathsf{st}_{\mathcal{I}}, \{\sigma_j\}_{j\in[\ell]})$. $\Pi\text{-}\mathsf{Execute}$ is a protocol between the client and the issuer (which also involves $\mathcal{E}$) with public inputs: a session identifier sid and the number of requested tokens ℓ, the client's private internal state $\mathsf{st}_{\mathcal{C}}$, and the issuer's

Game $\mathcal{G}^{\mathsf{BAT},\mathcal{E}}_{\mathsf{cor}}\big(1^\lambda, \ell_{\mathrm{s}}, \ell, \{\mathsf{ad}_j\}_{j\in[\ell_{\mathrm{s}}]}\big)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}_{\mathrm{iss}}, \mathsf{pk}_{\mathrm{iss}}) \leftarrow \mathsf{IssKGen}(\mathsf{pp})$
require $\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) = 1$
$(\mathsf{sid}, \mathsf{st}_{\mathcal{C}}, \mathsf{st}_{\mathcal{I}}) \leftarrow \Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathrm{iss}}, \ell, \mathsf{sk}_{\mathrm{iss}})$
$\{\sigma_j\}_{j\in[\ell]} \leftarrow \Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}, \ell, \mathsf{st}_{\mathcal{C}}, \mathsf{st}_{\mathcal{I}})$
for $j \in [\ell_{\mathrm{s}}]$:
  $\rho_j \leftarrow \mathsf{TokGen}(\sigma_j, \mathsf{ad}_j)$
  require $\mathsf{Spend}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \mathsf{ad}_j, \rho_j) = 1$
require $\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid}, \{\sigma_j\}_{j\in(\ell_{\mathrm{s}},\ell]}, \mathsf{st}_{\mathcal{C}}) \neq 0$
return 1

Figure 2: *Correctness game* $\mathcal{G}^{\mathsf{BAT},\mathcal{E}}_{\mathsf{cor}}$ *for a blockchain anonymous token protocol* BAT.

private internal state $\mathsf{st}_{\mathcal{I}}$. Upon successful execution, the protocol privately outputs ℓ *pre-tokens* $\{\sigma_j\}_{j\in[\ell]}$ to the client, and burns ℓ ETH from its account, and outputs the issuer's modified state $\mathsf{st}_{\mathcal{I}}$. Otherwise, it outputs $\perp$.

- $\mathsf{TokGen}(\sigma, \mathsf{ad}) \to \rho$. A client uses the randomized token generation algorithm on input a pre-token σ and associated data $\mathsf{ad} \in \{0,1\}^*$ it wants to associate the token σ to generate an *anonymous* token ρ.
- $\mathsf{Spend}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \mathsf{ad}, \rho) \to 0/1$. Any client uses the spend interface to consume an anonymous token. The interface takes as input an issuer's public key $\mathsf{pk}_{\mathrm{iss}}$, some associated data $\mathsf{ad} \in \{0,1\}^*$, and a token ρ. Upon successful execution, $\mathcal{E}$ mints 1 unit of ETH to be used by the associated data ad and outputs 1. Otherwise, it outputs 0.
- $\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid}, \{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}]}, \mathsf{st}_{\mathcal{C}}) \to 0/1$. Any client with its internal state $\mathsf{st}_{\mathcal{C}}$, uses the refund protocol to get a refund for a set of ℓ_{ref} pre-tokens $\{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}]}$ associated with an issuance session sid. Upon successful execution of this protocol, $\mathcal{E}$ mints ℓ_{ref} ETH for the client and outputs 1. It outputs 0 otherwise.

4.2. Required Properties

We describe the properties a BAT protocol must satisfy.

We begin with the standard notion of *correctness* that formalizes the behavior of the protocol when all parties act honestly. We capture correctness using the game $\mathcal{G}^{\mathsf{BAT},\mathcal{E}}_{\mathsf{cor}}$ in Fig. 2. In all our figures, we use the notation "**require** ⟨condition⟩" as a shorthand for "if condition is false, then return $\perp$, otherwise continue".

Definition 4.1 (Correctness)**.** A blockchain anonymous token protocol BAT is correct if for all $\ell_{\mathrm{s}}, \ell \in \mathbb{N}$ with $\ell_{\mathrm{s}} \leq \ell$, all associated data $\{\mathsf{ad}_j\}_{j\in[\ell]} \in \{0,1\}^*$,

$$\Pr\left[\mathcal{G}^{\mathsf{BAT},\mathcal{E}}_{\mathsf{cor}}\big(1^\lambda, \ell_{\mathrm{s}}, \ell, \{\mathsf{ad}_j\}_{j\in[\ell_{\mathrm{s}}]}\big) \Rightarrow 1\right] = 1$$

Next, we define *issuer fairness* which ensures that an honest issuer's balance remains non-negative despite adversarial behavior by clients. In other words, we require that total number of tokens N_{ex} issued by the issuer is always

Game $\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{issuer\text{-}fair}}\big(1^\lambda\big)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}_{\mathrm{iss}}, \mathsf{pk}_{\mathrm{iss}}) \leftarrow \mathsf{IssKGen}(\mathsf{pp})$
$\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}})$
$\mathsf{st}_{\mathcal{I}} \coloneqq \varnothing$
$N_{\mathsf{ex}} \coloneqq 0$; $N_{\mathsf{spd}} \coloneqq 0$; $N_{\mathsf{ref}} \coloneqq 0$
$\mathsf{done} \leftarrow \mathcal{A}^{\mathcal{E},\mathcal{O}\text{-}\mathsf{MIssue},\mathcal{O}\text{-}\mathsf{Execute},\mathcal{O}\text{-}\mathsf{Refund},\mathcal{O}\text{-}\mathsf{Spend}}(\cdot)$
return $N_{\mathsf{ex}} < N_{\mathsf{spd}} + N_{\mathsf{ref}}$

Oracle $\mathcal{O}$-MIssue(ℓ):

$(\mathsf{sid}, \mathsf{st}_{\mathcal{I}}, \star) \leftarrow \Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathrm{iss}}, \ell, \mathsf{sk}_{\mathrm{iss}})$

Oracle $\mathcal{O}$-Execute$\big(\mathsf{sid}, \ell\big)$:

if $\Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}, \ell, \mathsf{st}_{\mathcal{I}}, \star) \neq \perp$:
  $N_{\mathsf{ex}} \coloneqq N_{\mathsf{ex}} + \ell$

Oracle $\mathcal{O}$-Spend(ad, ρ):

if $\mathsf{Spend}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \rho, \mathsf{ad}) = 1$:
  $N_{\mathsf{spd}} \coloneqq N_{\mathsf{spd}} + 1$

Oracle $\mathcal{O}$-Refund$\big(\mathsf{sid}, \{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}]}, \mathsf{st}_{\mathcal{C}}\big)$:

if $\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid}, \{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}]}, \mathsf{st}_{\mathcal{C}}) = 1$:
  $N_{\mathsf{ref}} \coloneqq N_{\mathsf{ref}} + \ell_{\mathrm{ref}}$

Figure 3: *Issuer fairness game* $\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{issuer\text{-}fair}}$ *for a blockchain anonymous token protocol* BAT.

greater than or equal to the sum $N_{\mathsf{spd}} + N_{\mathsf{ref}}$, where N_{spd} is the total number of spent tokens issued under the issuer's key and N_{ref} is the total amount refunded by the issuer.

Definition 4.2 (Issuer Fairness)**.** A blockchain anonymous token protocol BAT ensures issuer fairness if for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{issuer\text{-}fair}}\big(1^\lambda\big) \Rightarrow 1\right] = \mathsf{negl}(\lambda)$$

We also define the complementary notion of *client fairness* which ensures that an honest client who pays for ℓ tokens can either spend up to ℓ tokens (with respect to any associated data $\mathsf{ad}_j \in \{0,1\}^*$ of its choice) or receive a full refund for the remaining unspent tokens. More formally:

Definition 4.3 (Client Fairness)**.** A blockchain anonymous token protocol BAT ensures client fairness if for all PPT adversary $\mathcal{A}$, for all $\ell \in \mathbb{N}$ and all $\{\mathsf{ad}_j\}_{j\in[\ell]} \in \{0,1\}^*$,

$$\Pr\left[\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{client\text{-}fair}}\big(1^\lambda, \ell, \{\mathsf{ad}_j\}_{j\in[\ell]}\big) \Rightarrow 1\right] = \mathsf{negl}(\lambda)$$

The *unlinkability* property captures the core notion of *anonymity* in our protocol. At a high level, unlinkability guarantees that an adversary, controlling the issuer, cannot link the final token to its corresponding issuance. The formal definition is intuitively similar to the corresponding notion of blindness in two-move blind signatures, extended to the batched setting. In particular, the adversary engages in two sessions of issuance protocols and is later shown finalized tokens in a random order. It must then guess which issuance

Game $\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{client\text{-}fair}}(1^\lambda, \ell, \{\mathsf{ad}_j\}_{j\in[\ell]})$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) \leftarrow \mathcal{A}(\mathsf{pp})$
require $\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) = 1$
$(\mathsf{sid}, \star, \mathsf{st}_{\mathcal{C}}) \leftarrow \Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathrm{iss}}, \ell, \star)$
$(\star, \{\sigma_j\}_{j\in[\ell]}) \leftarrow \Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}, \ell, \star, \mathsf{st}_{\mathcal{C}})$
for $j \in [\ell]$:
  $\rho_j \leftarrow \mathsf{TokGen}(\sigma_j, \mathsf{ad}_j)$
  if $\mathsf{Spend}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \mathsf{ad}_j, \rho_j) = 0$:
    require $\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid}, \{\sigma_k\}_{k\in[j,\ell]}, \mathsf{st}_{\mathcal{C}}) = 0$
    return 1
return 0

Figure 4: *The client fairness game* $\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{client\text{-}fair}}$ *for a blockchain anonymous token protocol* BAT.

session produced which token. Crucially, this guarantee holds for any two tokens or pairs of tokens, regardless of whether they were issued within the same batch (identified by a common sid) or across different batches. For the adversary, tokens issued in the same batch and tokens issued in different batches should be indistinguishable.

Definition 4.4 (Unlinkability)**.** A blockchain anonymous token protocol BAT is unlinkable if for all PPT adversaries $\mathcal{A}$, and all $\ell_0, \ell_1 \in \mathbb{N}$,

$$\left|\Pr\left[\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{unlink}}(1^\lambda, \ell_0, \ell_1) \Rightarrow 1\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda)$$

Finally, *solvency* prevents malicious issuers from minting and redeeming anonymous tokens without paying for them. Recall that we consider a single untrusted issuer who issues tokens to clients. This is unlike threshold issuance of anonymous tokens where a committee of n issuers with up to t corrupt issuers collective issue anonymous tokens. In a threshold issuance protocol, we can assume that all token issuances are recorded with the $\mathcal{E}$. Contrary to threshold issuance, in a single issuer system like ours, a malicious issuer can issue unbounded number of anonymous tokens without ever reporting them to $\mathcal{E}$. The solvency property we define next prevents such attacks. More formally,

Definition 4.5 (Solvency)**.** A blockchain anonymous token protocol BAT is solvent if for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\mathcal{G}^{\mathsf{BAT},\mathcal{E}}_{\mathsf{sol}}(1^\lambda) \Rightarrow 1\right] = \mathsf{negl}(\lambda)$$

Efficiency requirements. In addition to the above properties, a BAT scheme must satisfy the following practical efficiency requirements: (i) For an issuance session involving ℓ tokens, the on-chain computation and communication must be sublinear in ℓ (ideally $O(1)$). (ii) The issuer's off-chain issuance cost should be comparable to digitally signing ℓ short messages. (iii) The on-chain cost of verifying an anonymous token should be comparable to verifying standard digital signatures.

Game $\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{unlink}}(1^\lambda, \ell_0, \ell_1)$:

$b \leftarrow\$ \{0,1\}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) \leftarrow \mathcal{A}(\mathsf{pp})$
require $\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) = 1$
$(\mathsf{sid}_0, \star, \mathsf{st}_{\mathcal{C},0}) \leftarrow \Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathrm{iss}}, \ell_0, \star, \star)$
$(\star, \{\sigma_{0,j}\}_{j\in[\ell_0]}) \leftarrow \Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}_0, \ell_0, \star, \mathsf{st}_{\mathcal{C},0})$
$(\mathsf{sid}_1, \star, \mathsf{st}_{\mathcal{C},1}) \leftarrow \Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathrm{iss}}, \ell_1, \star, \star)$
$(\star, \{\sigma_{1,j}\}_{j\in[\ell_0]}) \leftarrow \Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}_1, \ell_1, \star, \mathsf{st}_{\mathcal{C},1})$
$\tilde{\ell}_0 \coloneqq 0, \quad \tilde{\ell}_1 \coloneqq 0$
$b' \leftarrow \mathcal{A}^{\mathcal{O}\text{-}\mathsf{Token},\mathcal{O}\text{-}\mathsf{Chal}}(\cdot)$
return $b = b'$

Oracle $\mathcal{O}\text{-}\mathsf{Token}(\tilde{b}, \mathsf{ad})$:

require $(\tilde{\ell}_{\tilde{b}} < \ell_{\tilde{b}})$
$\tilde{\ell}_{\tilde{b}} \coloneqq \tilde{\ell}_{\tilde{b}} + 1$
return $\mathsf{TokGen}(\sigma_{\tilde{b},\tilde{\ell}_{\tilde{b}}}, \mathsf{ad})$

Oracle $\mathcal{O}\text{-}\mathsf{Chal}(\mathsf{ad}_0, \mathsf{ad}_1)$:

require $(\tilde{\ell}_0 < \ell_0) \wedge (\tilde{\ell}_1 < \ell_1)$
$\tilde{\ell}_0 \coloneqq \tilde{\ell}_0 + 1; \quad \tilde{\ell}_1 \coloneqq \tilde{\ell}_1 + 1$
$\hat{\rho}_0 \coloneqq \mathsf{TokGen}(\sigma_{b,\tilde{\ell}_b}, \mathsf{ad}_0)$
$\hat{\rho}_1 \coloneqq \mathsf{TokGen}(\sigma_{1-b,\tilde{\ell}_{1-b}}, \mathsf{ad}_1)$
if $\hat{\rho}_0 = \perp$ or $\hat{\rho}_1 = \perp$: **return** $(\perp, \perp)$
return $(\hat{\rho}_0, \hat{\rho}_1)$

Figure 5: *Unlinkability game* $\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{unlink}}$ *for a blockchain anonymous token protocol* BAT.

5. Design

We summarize our protocol in Fig. 7 and describe it next.

Setup and key generation. The public parameters consists of description of a bilinear pairing group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, \circ) \leftarrow \mathsf{GGen}(1^\lambda)$, and public parameters $\mathsf{pp}_{\mathsf{DS}}$ of a digital signature scheme DS. We also use a hash function $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{G}_1$ modeled as a random oracle. The public parameters additionally include other constants: T_{ret} representing the minimum duration between issuer registration and retirement, and T_{grace} determining when issuance is paused near retirement, with $T_{\mathsf{grace}} \geq 2\delta_{\mathsf{inc}}$(cf. Section 4).

Our protocol uses the blind BLS signature as anonymous tokens [71]. Thus the private issuing key of an issuer is the BLS secret key $\mathsf{sk}_{\mathrm{iss}} \leftarrow\$ \mathbb{Z}_p$ and the associated public key is $\mathsf{pk}_{\mathrm{iss}} := [\![\mathsf{sk}_{\mathrm{iss}}]\!]_2$. Clients generate a key pair $(\mathsf{sk}_{\mathrm{ref}}, \mathsf{pk}_{\mathrm{ref}}) \leftarrow \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}})$ to authorize and receive refunds.

Issuer registration. Any issuer $\mathcal{I}$ registers its public issuing key $\mathsf{pk}_{\mathrm{iss}}$ with $\mathcal{E}$ using the $\mathsf{Register}^{\mathcal{E}}$ interface by depositing ℓ units of ETH as collateral. The registration is successful if $\mathcal{I}$ has sufficient balance and has not already registered. Upon successful registration, $\mathcal{E}$ locks ℓ_{iss} ETH from the $\mathcal{I}$'s balance into an escrow account $\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathrm{iss}}]$, sets the retirement

Game $\mathcal{G}^{\mathsf{BAT},\mathcal{E}}_{\mathsf{sol}}(1^\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) \leftarrow \mathcal{A}(\mathsf{pp})$
require $\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) = 1$
$t := \ell_{\mathrm{iss}}$
$\mathsf{done} \leftarrow \mathcal{A}^{\mathcal{E},\mathcal{O}\text{-}\mathsf{Execute},\mathcal{O}\text{-}\mathsf{Refund},\mathcal{O}\text{-}\mathsf{Spend}}(\cdot)$
return $t < 0$

Oracle $\mathcal{O}\text{-}\mathsf{Execute}(\mathsf{sid}, \ell)$:

if $\Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}, \ell, \star, \star) \neq \bot$:
  $t := t + \ell$

Oracle $\mathcal{O}\text{-}\mathsf{Spend}(\mathsf{aux}, \rho)$:

if $\mathsf{Spend}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \rho, \mathsf{aux}) = 1$:
  $t := t - 1$

Oracle $\mathcal{O}\text{-}\mathsf{Refund}(\mathsf{sid}, \ell_r, \{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}]})$:

if $\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid}, \{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}]}, \star) = 1$:
  $t := t - \ell_{\mathrm{ref}}$

Figure 6: *Solvency game $\mathcal{G}^{\mathsf{BAT},\mathcal{E}}_{\mathsf{sol}}$ for a blockchain anonymous token protocol* BAT.

deadline $\mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathrm{iss}}] := \mathcal{E}.\mathsf{currentHeight} + T_{\mathsf{ret}}$, and initializes counters: $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathrm{iss}}] := 0$ to track the number of tokens issued by $\mathcal{I}$, $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathrm{iss}}] := 0$ to track tracks of spent tokens, and $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathrm{iss}}] := 0$ to track the number of refunded tokens. $\mathcal{I}$ also initializes a local variable $\mathsf{st}_{\mathcal{I}}.\mathsf{Niss} := 0$ to track the number of anonymous token it has issued so far.

Masked token issuance. $\Pi\text{-}\mathsf{MIssue}$ is a two move off-chain protocol between client $\mathcal{C}$ and issuer $\mathcal{I}$. At the end of this protocol, $\mathcal{C}$ gets ℓ masked anonymous tokens issued by $\mathcal{I}$.

In the first move, $\mathcal{C}$ generates ℓ ephemeral key pairs $\{(\mathsf{sk}_{\mathrm{eph},i}, \mathsf{pk}_{\mathrm{eph},i})\}_{i\in[\ell]}$ for the digital signature scheme DS. Next, for each $i \in [\ell]$, $\mathcal{C}$ samples a blinding factor $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes the blinded values $\boldsymbol{X} = \{X_i = r_i \cdot H(\mathsf{pk}_{\mathrm{eph},i})\}_{i\in[\ell]}$, and sends them to $\mathcal{I}$ along with a random session identifier sid.

In the second move, $\mathcal{I}$ upon receiving the blinded message from $\mathcal{C}$, checks that sid has not been used before. If so, $\mathcal{I}$ samples a fresh masking key $k \leftarrow \mathbb{Z}_p$ and computes its commitment $\mathsf{com}_k = k \cdot \mathsf{pk}_{\mathrm{iss}}$. Next, $\mathcal{I}$ computes blind signatures $\tilde{\sigma}_i = (k \cdot \mathsf{sk}_{\mathrm{iss}}) \cdot X_i$ for each $i \in [\ell]$. $\mathcal{I}$ then sends the masked blind signatures $(\{\tilde{\sigma}_i\}_{i\in[\ell]}, \mathsf{com}_k)$ to $\mathcal{C}$.

$\mathcal{C}$ upon receiving the masked blind signatures validates them by checking whether $\tilde{\sigma}_i \circ [\![1]\!]_2 = X_i \circ \mathsf{com}_k$ for each $i \in [\ell]$. $\mathcal{C}$ can batch verify all ℓ masked tokens by sampling $\beta_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and checking $\sum_{i\in[\ell]} \beta_i \cdot \tilde{\sigma}_i \circ [\![1]\!]_2 = \sum_{i\in[\ell]} \beta_i X_i \circ \mathsf{com}_k$. $\mathcal{C}$ also samples a key pair for potential refunds later.

On-chain atomic issuance. The $\Pi\text{-}\mathsf{Execute}$ protocol coordinates the masking key revelation and payment for it via the blockchain $\mathcal{E}$. The client $\mathcal{C}$ submits $(\mathsf{sid}, \mathsf{pk}_{\mathrm{ref}}, \mathsf{com}_k, \ell)$ to $\mathcal{E}$, where $\mathsf{pk}_{\mathrm{ref}}$ is the refund public key, and escrows the payment for ℓ tokens. In response, $\mathcal{I}$ reveals the masking key k to $\mathcal{E}$, while ensuring it issues at most $\ell_{\mathrm{iss}}/2$ tokens.

Upon receiving k, $\mathcal{E}$ checks: (1) k matches the commitment, i.e., $k \cdot \mathsf{pk}_{\mathrm{iss}} = \mathsf{com}_k$, (2) the issuer has not retired, (3) issuance is not too close to retirement, i.e., $\mathcal{E}.\mathsf{currentHeight} + T_{\mathsf{grace}} \leq \mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathrm{iss}}]$, and (4) the issuer has sufficient collateral to back this issuance via $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathrm{iss}}] + \ell \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathrm{iss}}]/2$. The third check essentially freezes issuance near the retirement deadline so as to ensures that the client has sufficient time to successfully spend or refund the issued tokens. The fourth check implements our solvency condition, ensuring the issuer maintains at least twice the collateral as the total issued tokens. If all checks pass, $\mathcal{E}$ increments $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathrm{iss}}]$ by ℓ, burns ℓ coins from the $\mathcal{C}$'s balance, and stores $(\mathsf{pk}_{\mathrm{ref}}, \mathsf{com}_k, \ell)$ for future refunds.

Upon obtaining k, $\mathcal{C}$ recovers the BLS signature $\alpha_i = (r \cdot k)^{-1} \cdot \tilde{\sigma}_i$ on the i-th ephemeral public key, and stores the tuple $\sigma_i := (\alpha_i, \mathsf{pk}_{\mathrm{eph},i}, \mathsf{sk}_{\mathrm{eph},i})$ as the i-th pre-token.

Token generation. Given a pre-token $\sigma = (\alpha, \mathsf{sk}_{\mathrm{eph}}, \mathsf{pk}_{\mathrm{eph}})$, $\mathcal{C}$ binds it to some associated data ad by computing a digital signature $\gamma = \mathsf{DS.Sign}(\mathsf{sk}_{\mathrm{eph}}, \mathsf{ad})$. We define the tuple $\rho = (\mathsf{pk}_{\mathrm{eph}}, \alpha, \gamma)$ to be the anonymous token.

Token spending. To spend a token ρ, $\mathcal{C}$ posts $(\mathsf{pk}_{\mathrm{iss}}, \rho)$ to $\mathcal{E}$. The spend is successful if: (i) $\mathcal{I}$ (with public key $\mathsf{pk}_{\mathrm{iss}}$) has not retired, (ii) Number of spent tokens associated with $\mathcal{I}$ is less than ℓ_{iss}. (iii) α is a valid BLS signature on $\mathsf{pk}_{\mathrm{eph}}$ under the issuer $\mathsf{pk}_{\mathrm{iss}}$, i.e., $\alpha \circ [\![1]\!]_2 = H(\mathsf{pk}_{\mathrm{eph}}) \circ \mathsf{pk}_{\mathrm{iss}}$, (iv) γ is a valid signature on ad under $\mathsf{pk}_{\mathrm{eph}}$, and (5) $\mathsf{pk}_{\mathrm{eph}}$ has not been spent before, i.e., $\mathsf{pk}_{\mathrm{eph}} \notin \mathcal{E}.\mathsf{Nullifiers}$.

Upon successful spend, $\mathcal{E}$ adds $\mathsf{pk}_{\mathrm{eph}}$ to the nullifier set $\mathcal{E}.\mathsf{Nullifiers}$, increments $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathrm{iss}}]$, and mints 1 unit of ETH to be used by the associated data ad.

Refunds. To claim refunds of ℓ_{ref} unspent tokens, $\mathcal{C}$ authorizes the refund request by signing sid, and the unspent ephemeral public keys and their BLS signatures using $\mathsf{sk}_{\mathrm{ref}}$. Let $\gamma_{\mathrm{ref}} := \mathsf{DS.Sign}(\mathsf{sk}_{\mathrm{ref}}, (\mathsf{sid}, \{\alpha_i, \mathsf{pk}_{\mathrm{eph},i}\}_{i\in[\ell_{\mathrm{ref}}]}))$ be the digital signature. $\mathcal{C}$ submits the refund claim posting $(\mathsf{sid}, \{\alpha_i, \mathsf{pk}_{\mathrm{eph},i}\}_{i\in[\ell_{\mathrm{ref}}]}, \gamma_{\mathrm{ref}})$ to $\mathcal{E}$.

Upon receiving the refund claim, $\mathcal{E}$ validates it by checking: (i) the session sid has not been refunded before, (ii) $\ell_{\mathrm{ref}} \leq \ell$ where ℓ is the batch size from session sid; (iii) for each $i \in [\ell_{\mathrm{ref}}]$, α_i is a valid BLS signature, i.e., $\alpha_i \circ [\![1]\!]_2 = H(\mathsf{pk}_{\mathrm{eph},i}) \circ \mathsf{pk}_{\mathrm{iss}}$, and (iv) None of the ephemeral keys has been spent. If all checks pass, $\mathcal{E}$ mints ℓ_{ref} ETH coins to $\mathsf{pk}_{\mathrm{ref}}$, adds all ephemeral keys to the nullifier set, increments $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathrm{iss}}]$ by ℓ_{ref}, and marks session sid as refunded.

Remark. We note that as an optimization, it is possible to use the BLS aggregate signature $\alpha = \sum_{i=1}^{\ell_{\mathrm{ref}}} \alpha_i$ to prove the validity of all unspent tokens. However, proving this optimized protocol is issuer-fair requires additional techniques, which we leave for future work.

Issuer retirement. When an issuer $\mathcal{I}$ with public key $\mathsf{pk}_{\mathrm{iss}}$ submits a retirement request to recover its collateral, $\mathcal{E}$ first requires that the retirement deadline has been reached, i.e., $\mathcal{E}.\mathsf{currentHeight} \geq \mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathrm{iss}}]$. If so, $\mathcal{E}$ computes

Requirements: Digital signature DS, pairing groups $\mathcal{G} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, \circ)$ and a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$.

$\mathsf{Setup}(1^\lambda)$

1 : $\mathsf{pp}_{\mathsf{DS}} \leftarrow \mathsf{DS.Setup}(1^\lambda)$
2 : **return** $\mathsf{pp} := (\mathcal{G}, \mathsf{pp}_{\mathsf{DS}}, T_{\mathsf{ret}}, T_{\mathsf{grace}})$

$\mathsf{IssKGen}(\mathsf{pp})$

1 : $\mathsf{sk}_{\mathsf{iss}} \leftarrow\!\$\ \mathbb{Z}_p^*$; $\mathsf{pk}_{\mathsf{iss}} := [\![\mathsf{sk}_{\mathsf{iss}}]\!]_2$
2 : **return** $(\mathsf{sk}_{\mathsf{iss}}, \mathsf{pk}_{\mathsf{iss}})$

$\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathsf{iss}}, \ell_{\mathsf{iss}})$

1 : $\mathcal{I}$: **send** $\mathsf{pk}_{\mathsf{iss}}$ to $\mathcal{E}$
2 : $\mathcal{E}$: **require** $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathsf{iss}}] >= \ell_{\mathsf{iss}}$ and $\mathsf{pk}_{\mathsf{iss}} \notin \mathcal{E}.\mathsf{esc}$
3 : $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathsf{iss}}] := \mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathsf{iss}}] - \ell_{\mathsf{iss}}$
4 : $\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}] := \ell_{\mathsf{iss}}$; $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] := 0$
5 : $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}] := 0$; $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}] := 0$
6 : $\mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathsf{iss}}] := \mathcal{E}.\mathsf{currentHeight} + T_{\mathsf{ret}}$

$\mathsf{Retire}^{\mathcal{E}}(\mathsf{pk}_{\mathsf{iss}})$

1 : **require** $\mathcal{E}.\mathsf{currentHeight} \geq \mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathsf{iss}}]$
2 : $\Delta := \max(0, \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}] - \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}])$
3 : $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathsf{iss}}] := \mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}] - \Delta$ *// burn* Δ ETH
4 : $\mathcal{E}.\mathsf{retired} := \mathcal{E}.\mathsf{retired} \cup \{\mathsf{pk}_{\mathsf{iss}}\}$

$\Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathsf{iss}}, \ell, \mathsf{sk}_{\mathsf{iss}}, \mathsf{st}_{\mathcal{I}})$

$\mathcal{C}$: *// Generating blinded message*
1 : $\mathsf{sid} \leftarrow\!\$\ \{0,1\}^\lambda$ *// session id*
2 : $\{(\mathsf{sk}_{\mathrm{eph},i}, \mathsf{pk}_{\mathrm{eph},i}) \leftarrow \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}})\}_{i\in[\ell]}$
3 : $\boldsymbol{X} := \{r_i \cdot H(\mathsf{pk}_{\mathrm{eph},i})\}_{i\in[\ell]}$ where $r_i \leftarrow\!\$\ \mathbb{Z}_p$
4 : **send** $(\mathsf{sid}, \boldsymbol{X}, \ell)$ to $\mathcal{I}$
$\mathcal{I}$: *// Issuing masked blind signatures*
5 : **require** $\mathsf{sid} \notin \mathsf{st}_{\mathcal{I}}.\mathsf{sessions}$
6 : $k \leftarrow\!\$\ \mathbb{Z}_p^*$; $\mathsf{com}_k := k \cdot \mathsf{pk}_{\mathsf{iss}}$
7 : $\mathsf{st}_{\mathcal{I}}.\mathsf{sessions}[\mathsf{sid}] := (\mathsf{com}_k, \ell, k)$
8 : $\{\tilde{\sigma}_i := (k \cdot \mathsf{sk}_{\mathsf{iss}}) \cdot X_i\}_{i\in[\ell]}$
9 : **send** $(\{\tilde{\sigma}_i\}_{i\in[\ell]}, \mathsf{com}_k)$ to $\mathcal{C}$
$\mathcal{C}$: *// Verifying masked blind signatures*
10 : **require** $\forall i \in [\ell] : \tilde{\sigma}_i \circ [\![1]\!]_2 = X_i \circ \mathsf{com}_k$
11 : $(\mathsf{sk}_{\mathrm{ref}}, \mathsf{pk}_{\mathrm{ref}}) \leftarrow \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}})$
12 : $\mathsf{st}_{\mathcal{C}} := \left(\begin{array}{l} \{r_i, \tilde{\sigma}_i, \mathsf{sk}_{\mathrm{eph},i}, \mathsf{pk}_{\mathrm{eph},i}\}_{i\in[\ell]}, \\ \mathsf{sk}_{\mathrm{ref}}, \mathsf{pk}_{\mathrm{ref}}, \mathsf{com}_k \end{array} \right)$
13 : **return** $(\mathsf{sid}, \mathsf{st}_{\mathcal{C}})$

$\mathsf{TokGen}(\sigma, \mathsf{ad})$

1 : **parse** σ as $(\alpha, \mathsf{sk}_{\mathrm{eph}}, \mathsf{pk}_{\mathrm{eph}})$
2 : $\gamma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathrm{eph}}, \mathsf{ad})$
3 : **return** $\rho := (\mathsf{pk}_{\mathrm{eph}}, \alpha, \gamma)$

$\Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}, \ell, \mathsf{st}_{\mathcal{I}}, \mathsf{st}_{\mathcal{C}})$

$\mathcal{C}$: *// Represented by* pk_{c}
1 : **parse** $\mathsf{st}_{\mathcal{C}}$ as $(\star, \star, \mathsf{pk}_{\mathrm{ref}}, \mathsf{com}_k)$
2 : **send** $(\mathsf{sid}, \mathsf{pk}_{\mathrm{ref}}, \mathsf{com}_k, \ell)$ to $\mathcal{E}$
$\mathcal{I}$: *// Represented by* $\mathsf{pk}_{\mathrm{iss}}$
3 : **require** $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathrm{c}}] \geq \ell$ and $\mathsf{st}_{\mathcal{I}}.\mathsf{sessions}[\mathsf{sid}] = (\mathsf{com}_k, \ell, \star)$
4 : **require** $2(\mathsf{st}_{\mathcal{I}}.\mathsf{Niss} + \ell) \leq \ell_{\mathrm{iss}}$
5 : $\mathsf{st}_{\mathcal{I}}.\mathsf{Niss} := \mathsf{st}_{\mathcal{I}}.\mathsf{Niss} + \ell$ *// $\mathsf{st}_{\mathcal{I}}$.Niss initialized to 0*
6 : **parse** $\mathsf{st}_{\mathcal{I}}.\mathsf{sessions}[\mathsf{sid}]$ as $(\star, \star, k)$
7 : **send** (sid, k) to $\mathcal{E}$
8 : $\mathcal{E}$: **require** $\mathsf{pk}_{\mathrm{iss}} \notin \mathcal{E}.\mathsf{retired}$
9 : **require** $\mathcal{E}.\mathsf{currentHeight} + T_{\mathsf{grace}} \leq \mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathrm{iss}}]$
10 : **require** $k \cdot \mathsf{pk}_{\mathrm{iss}} = \mathsf{com}_k$
11 : **require** $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathrm{iss}}] + \ell \leq \frac{\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathrm{iss}}]}{2}$
12 : $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathrm{iss}}] := \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathrm{iss}}] + \ell$
13 : $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathrm{c}}] := \mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathrm{c}}] - \ell$ *// Burn* ℓ ETH
14 : $\mathcal{E}.\mathsf{sessions}[\mathsf{sid}] := (\mathsf{pk}_{\mathrm{ref}}, \mathsf{com}_k, \ell)$
$\mathcal{C}$: *// Use k to unmask pre-tokens*
15 : **parse** $\mathsf{st}_{\mathcal{C}}$ as $(\{r_i, \tilde{\sigma}_i, \mathsf{sk}_{\mathrm{eph},i}, \mathsf{pk}_{\mathrm{eph},i}\}_{i\in[\ell]}, \star, \star, \star)$
16 : **return** $\{\sigma_i := ((r_i \cdot k)^{-1} \cdot \tilde{\sigma}_i, \mathsf{sk}_{\mathrm{eph},i}, \mathsf{pk}_{\mathrm{eph},i})\}_{i\in[\ell]}$

$\mathsf{Spend}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \mathsf{ad}, \rho)$

1 : $\mathcal{C}$: **send** $(\mathsf{pk}_{\mathrm{iss}}, \mathsf{ad}, \rho)$ to $\mathcal{E}$
2 : $\mathcal{E}$: **require** $\mathsf{pk}_{\mathrm{iss}} \notin \mathcal{E}.\mathsf{retired}$
3 : **require** $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathrm{iss}}] < \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathrm{iss}}]$
4 : **parse** ρ as $(\mathsf{pk}_{\mathrm{eph}}, \alpha, \gamma)$
5 : **require** $\alpha \circ [\![1]\!]_2 = \mathsf{H}(\mathsf{pk}_{\mathrm{eph}}) \circ \mathsf{pk}_{\mathrm{iss}}$ *// Issuer's signature*
6 : **require** $\mathsf{pk}_{\mathrm{eph}} \notin \mathcal{E}.\mathsf{Nullifiers}$
7 : **require** $\mathsf{DS.Verify}(\mathsf{pk}_{\mathrm{eph}}, \mathsf{ad}, \gamma) = 1$
8 : $\mathcal{E}.\mathsf{Nullifiers} := \mathcal{E}.\mathsf{Nullifiers} \cup \{\mathsf{pk}_{\mathrm{eph}}\}$
9 : $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathrm{iss}}] := \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathrm{iss}}] + 1$ *// Mint 1* ETH

$\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid}, \{\sigma_i\}_{i\in[\ell_{\mathrm{ref}}]}, \mathsf{st}_{\mathcal{C}})$

1 : $\mathcal{C}$: **parse** σ_i as $(\alpha_i, \star, \mathsf{pk}_{\mathrm{eph},i})$ for $i \in [\ell_{\mathrm{ref}}]$
2 : $\gamma_{\mathrm{ref}} \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathrm{ref}}, (\mathsf{sid}, \{\alpha_i, \mathsf{pk}_{\mathrm{eph},i}\}_{i\in[\ell_{\mathrm{ref}}]}))$
3 : **send** $(\mathsf{sid}, \{\alpha_i, \mathsf{pk}_{\mathrm{eph},i}\}_{i\in[\ell_{\mathrm{ref}}]}, \gamma_{\mathrm{ref}})$ to $\mathcal{E}$
4 : $\mathcal{E}$: **require** $\mathsf{DS.Verify}(\mathsf{pk}_{\mathrm{ref}}, (\mathsf{sid}, \{\alpha_i, \mathsf{pk}_{\mathrm{eph},i}\}_{i\in[\ell_{\mathrm{ref}}]}), \gamma_{\mathrm{ref}})$
5 : **require** $\mathsf{sid} \in \mathcal{E}.\mathsf{sessions}$ and $\mathsf{sid} \notin \mathcal{E}.\mathsf{refundset}$
6 : **parse** $\mathcal{E}.\mathsf{sessions}[\mathsf{sid}]$ as $(\mathsf{pk}_{\mathrm{ref}}, \star, \ell)$
7 : **require** $\ell_{\mathrm{ref}} \leq \ell$ and $\mathsf{pk}_{\mathrm{iss}} \notin \mathcal{E}.\mathsf{retired}$
8 : **for** $i \in [\ell_{\mathrm{ref}}]$, **require** $\alpha_i \circ [\![1]\!]_2 = \mathsf{H}(\mathsf{pk}_{\mathrm{eph},i}) \circ \mathsf{pk}_{\mathrm{iss}}$
9 : **for** $i \in [\ell_{\mathrm{ref}}]$: **require** $\mathsf{pk}_{\mathrm{eph},i} \notin \mathcal{E}.\mathsf{Nullifiers}$
10 : $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathrm{ref}}] := \mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\mathrm{ref}}] + \ell_{\mathrm{ref}}$ *// Mint* ℓ_{ref} ETH
11 : $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathrm{iss}}] := \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathrm{iss}}] + \ell_{\mathrm{ref}}$
12 : $\mathcal{E}.\mathsf{refundset} := \mathcal{E}.\mathsf{refundset} \cup \{\mathsf{sid}\}$
13 : $\mathcal{E}.\mathsf{Nullifiers} := \mathcal{E}.\mathsf{Nullifiers} \cup \{\mathsf{pk}_{\mathrm{eph},i} | i \in [\ell_{\mathrm{ref}}]\}$

Figure 7: BAT *protocol with compact issuance. $\mathcal{C}$: actions by the client, $\mathcal{I}$: by the issuer, $\mathcal{E}$: by the environment.*

$\Delta = \max(0, \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}] - \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}])$, which captures any deficit due to $\mathcal{I}$ maliciously issuing tokens without an on-chain exchange. $\mathcal{E}$ then returns $\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}] - \Delta$ units of ETH (after deducting some penalty) to the $\mathcal{I}$, burns Δ units of ETH, and marks $\mathsf{pk}_{\mathsf{iss}}$ as retired.

Towards non-interactive refunds. In Fig. 8, we present an alternative BAT scheme with a different on-chain/off-chain cost tradeoff. In particular, this scheme enables refunding ℓ_r tokens using only $\mathcal{O}(1)$ on-chain data and provides provisions for non-interactive refunds, but at the cost of increased off-chain costs for the client.

In this scheme, during Π-MIssue, $\mathcal{C}$ generates the ephemeral keys using a pseudorandom function (PRF) keyed with a random seed. Additionally, $\mathcal{C}$ provides a zero-knowledge proof to $\mathcal{I}$ proving that the blinded values correspond to ephemeral keys correctly derived using a PRF on the seed. $\mathcal{I}$ verifies this proof and proceeds with the masked issuance protocol as before. During Refund, $\mathcal{E}$ only requires the seed. In particular, $\mathcal{E}$ uses this seed and the PRF to deterministically regenerate the corresponding ephemeral keys and verify that the tokens are valid and unspent.

Although this scheme still requires $\mathcal{O}(\ell_{\mathrm{ref}})$ on-chain computation, it facilitates efficient automated refunds. A client seeking automated refunds can store the encryption of the seed on-chain using off-the-shelf time-lock encryption [72], [73] at issuance time. The seed can later be automatically decrypted and processed to issue a refund to the client without any client interaction. We also implement this variant and report benchmarks in Section 7.

6. Analysis of BAT

Lemma 6.1 (Correctness)**.** *The* BAT *protocol in Fig. 7 satisfies correctness as per Def. 4.1.*

Proof. To argue correctness, we assume that the issuer $\mathcal{I}$ escrows sufficient funds during registration. Recall that each masked pre-token in our scheme is of the form $\tilde{\sigma}_i = (k \cdot \mathsf{sk}_{\mathsf{iss}}) \cdot X_i$ and $\mathsf{com}_k = k \cdot \mathsf{pk}_{\mathsf{iss}}$. Thus, we get

$$\tilde{\sigma}_i \circ [\![1]\!]_2 = (k \cdot \mathsf{sk}_{\mathsf{iss}}) \cdot X_i \circ \cdot [\![1]\!]_2 = X_i \circ k \cdot \mathsf{pk}_{\mathsf{iss}}.$$

Therefore, the Π-MIssue protocol will successful. Moreover, once the client $\mathcal{C}$ obtains k as part of Π-Execute, it can unmask the pre-token by computing $\alpha_i := (r \cdot k)^{-1} \cdot \tilde{\sigma}_i = \mathsf{sk}_{\mathsf{iss}} \cdot X_i$, which is a BLS signature on $\mathsf{pk}_{\mathrm{eph},i}$. Finally, the correctness of the Spend and Refund follows from the correctness of the underlying signature scheme DS. □

Lemma 6.2 (Client Fairness)**.** *The* BAT *protocol in Fig. 7 satisfies client-fairness as per Def. 4.3.*

Proof. To prove client fairness, we argue that if the Π-MIssue and Π-Execute protocols complete successfully, then the client $\mathcal{C}$ can either spend all issued tokens or obtain refunds for any unspent tokens.

During Π-MIssue, the validity check $\tilde{\sigma}_i \circ [\![1]\!]_2 = X_i \circ \mathsf{com}_k$ implies that $\tilde{\sigma}_i = (k \cdot \mathsf{sk}_{\mathsf{iss}}) \cdot X_i$ where $\mathsf{com}_k = k \cdot \mathsf{pk}_{\mathsf{iss}}$. Upon successful completion of Π-Execute, $\mathcal{C}$ learns the masking key k. Using its knowledge of (r, k), $\mathcal{C}$ can unmask each pre-token by computing $\alpha_i := (r \cdot k)^{-1} \cdot \tilde{\sigma}_i = \mathsf{sk}_{\mathsf{iss}} \cdot X_i$, which is a BLS signature on $\mathsf{pk}_{\mathrm{eph},i}$.

This guarantees that $\mathcal{C}$ can always spend the pre-token σ_i, unless the spend limit for the issuer public key $\mathsf{pk}_{\mathsf{iss}}$ has already been reached. Furthermore, since spending a token associated with an ephemeral public key $\mathsf{pk}_{\mathrm{eph}}$ requires a digital signature under $\mathsf{pk}_{\mathrm{eph}}$, no adversary can steal and spend the token without violating the unforgeability of the underlying digital signature scheme.

If the spend capacity for $\mathsf{pk}_{\mathsf{iss}}$ is exhausted, $\mathcal{C}$ can request refunds for its unspent tokens. We note that, there is no bound on the number of tokens that may be refunded, provided that the refund request correctly indicates the corresponding issuance. In addition, the refund protocol returns funds to $\mathsf{pk}_{\mathrm{ref}}$, the public key selected by $\mathcal{C}$ at issuance.

It remains to show that a malicious issuer cannot retire before $\mathcal{C}$ has a chance to spend or refund the issued tokens. Issuance only succeeds in Π-Execute whenever $\mathcal{E}.\mathsf{currentHeight} + T_{\mathsf{grace}} \leq \mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathsf{iss}}]$. Under the bounded-delay assumption of Section 4, an honest transaction submitted by $\mathcal{C}$ is included within δ_{inc} blocks. Since $T_{\mathsf{grace}} \geq 2\delta_{\mathsf{inc}}$, $\mathcal{C}$ has time to (i) submit a Spend transaction and have it included, and, if that Spend fails because the (malicious) issuer's spend cap was reached, (ii) submit a Refund transaction for any remaining unspent tokens before the issuer retires at $\mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\mathsf{iss}}]$. □

Lemma 6.3 (Solvency)**.** *The* BAT *protocol in Fig. 7 is solvent as per Def. 4.5.*

Proof. To establish solvency, we show that an adversary can never mint more ETH than is burned, even if it controls all issuers and all clients.

For any issuer with public key $\mathsf{pk}_{\mathsf{iss}}$, our protocol mints ETH in exactly two cases: (i) each successful spend mints one unit of ETH; and (ii) each refund mints $\ell_r \leq \ell$ units of ETH, where ℓ is the number of tokens issued in the corresponding issuance session. Therefore, for issuer $\mathsf{pk}_{\mathsf{iss}}$, we mint $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}]$ units of ETH.

Our protocol also burns ETH in two cases. First, we burn ℓ units of ETH for issuing ℓ tokens. Second, at retirement we burn Δ units of ETH, where $\Delta := \max(0, \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}] - \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}])$.

The quantity Δ above precisely captures the excess ETH that the protocol mints on behalf of issuer $\mathsf{pk}_{\mathsf{iss}}$ beyond what is burnt during on-chain issuance associated with $\mathsf{pk}_{\mathsf{iss}}$. Therefore, by burning Δ units of ETH at retirement, the protocol maintains solvency.

We now argue that the protocol can always burn Δ units of ETH when retiring issuer $\mathsf{pk}_{\mathsf{iss}}$. Note that in our protocol: (i) for issuer $\mathsf{pk}_{\mathsf{iss}}$, we issue at most half of its escrowed amount, i.e., $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}]/2$; (ii) we allow spends of at most $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}]$ spends, i.e., $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] \leq \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}]/2$; and (iii) we allow refunds only for issued tokens, i.e., $\mathcal{E}.\mathsf{Refund}[\mathsf{pk}_{\mathsf{iss}}] \leq \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}]/2$. Combining these bounds, we obtain $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{Refund}[\mathsf{pk}_{\mathsf{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}]$. Since $\Delta \leq \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{Refund}[\mathsf{pk}_{\mathsf{iss}}]$, and the issuer $\mathsf{pk}_{\mathsf{iss}}$

escrows $\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}]$ units of ETH, we can always burn Δ units of ETH while retiring $\mathsf{pk}_{\mathsf{iss}}$. □

Lemma 6.4 (Unlinkability)**.** *The* BAT *protocol in Fig. 7 is unlinkable as per Def. 4.4.*

Proof. We prove that our protocol achieves information-theoretic unlinkability.

Consider the adversary $\mathcal{A}$'s view in the unlinkability game $\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{unlink}}$ (see Fig. 5). First, note that we mask each in a session independent of other tokens. Therefore, it suffices to analyze $\mathcal{A}$'s view of the challenge token.

Let $\mathsf{pk}_{\mathrm{eph}}$ and $\mathsf{pk}'_{\mathrm{eph}}$ denote the ephemeral public keys underlying the challenge token from sessions sid_0 and sid_1, respectively. During the issuance phase, $\mathcal{A}$ observes the blinded ephemeral keys: $Y_0 := r \cdot \mathsf{H}(\mathsf{pk}_{\mathrm{eph}})$ and $Y_1 := r' \cdot \mathsf{H}(\mathsf{pk}'_{\mathrm{eph}})$, for uniformly random and independent blinding factors $r, r' \leftarrow\$ \mathbb{Z}_p$. This implies that the blinded values Y_0 and Y_1 are distributed uniformly and independently in $\mathbb{G}$.

Now, for the pair of ephemeral keys $(\mathsf{pk}_{\mathrm{eph}}, \mathsf{pk}'_{\mathrm{eph}})$ and the group elements $(Y_0, Y_1) \in \mathbb{G}^2$ observed by $\mathcal{A}$, there exist uniformly random and independent blinding factors $s, s' \in \mathbb{Z}_p$ such that:

$$Y_0 = s \cdot \mathsf{H}(\mathsf{pk}'_{\mathrm{eph}}) \quad \text{and} \quad Y_1 = s' \cdot \mathsf{H}(\mathsf{pk}_{\mathrm{eph}})$$

Thus, $\mathcal{A}$'s view is equally consistent with either ordering of the challenge tokens. Therefore, we get:

$$\Pr[\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{unlink}}(1^\lambda, \ell_0, \ell_1) \Rightarrow 1] = 1/2. \qquad \square$$

Next, we prove that our protocol satisfies issuer fairness against any adversary in the Generic Group Model [74].

Theorem 6.5 (Issuer Fairness)**.** *Assume that* $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$ *is modeled as a random oracle into* $\mathbb{G}_1$*, and that* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, \circ)$ *is a type-3 generic bilinear group. For any adversary* $\mathcal{A}$ *that makes at most* q_I *issuance-session queries and creates at most* N_h *generic-group handles, the* BAT *protocol in Fig. 7 satisfies*

$$\Pr[\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{issuer\text{-}fair}}(1^\lambda) \Rightarrow 1] \le \frac{(2q_I+3)N_h^2}{p-1}.$$

In particular, BAT *ensures issuer-fairness as per Def. 4.2 in the generic group model (GGM).*

Proof. Let $x \in \mathbb{Z}_p^\star$ be the issuer secret key, so $\mathsf{pk}_{\mathsf{iss}} = [\![x]\!]_2$. Let X be the formal variable corresponding to x. For every distinct random-oracle input m_t, the random oracle returns a fresh $\mathbb{G}_1$ handle $[\![H_t]\!]_1$, where H_t is a new formal variable. For every issuance session sid, let K_{sid} be the formal variable corresponding to the masking key k_{sid} sampled by the issuer. Let N_h be an upper bound on the total number of generic-group handles created in the execution.

We work over the polynomial ring

$$\mathcal{R} := \mathbb{Z}_p\left[X, \{H_t\}_{t\in[q_H]}, \{K_{\mathsf{sid}}\}_{\mathsf{sid}\in[q_I]}\right],$$

where q_H is the total number of distinct random-oracle inputs made by $\mathcal{A}$ or internally by Spend and Refund. The initial handles satisfy $[\![1]\!]_1 \leftrightarrow 1$, $[\![1]\!]_2 \leftrightarrow 1$, and $\mathsf{pk}_{\mathsf{iss}} = [\![x]\!]_2 \leftrightarrow X$. Generic operations in each source group are $\mathbb{Z}_p$-linear operations on the corresponding polynomials, and a pairing query on source-group polynomials A and B returns the target-group polynomial AB. This is the standard bilinear GGM execution.

Issuance. Consider an issuance session sid for ℓ_{sid} tokens. If $\mathcal{A}$ submits blinded messages represented by $\mathbb{G}_1$ polynomials $P_{\mathsf{sid},1}, \ldots, P_{\mathsf{sid},\ell_{\mathsf{sid}}}$, the issuer returns $\tilde{\sigma}_{\mathsf{sid},i} = [\![K_{\mathsf{sid}} X P_{\mathsf{sid},i}]\!]_1$ and $\mathsf{com}_k = [\![K_{\mathsf{sid}} X]\!]_2$ for a fresh variable K_{sid}. If sid is later executed, $\mathcal{E}$ reveals k_{sid} and increments N_{ex} by ℓ_{sid}.

Collision event. Let Coll be the event that two distinct formal polynomials assigned to handles in the same group, and still distinct in the current partially substituted ring, evaluate to the same group element at some point in the execution. Here the current ring substitutes every masking key that has already been revealed and keeps the other masking keys formal. Every $\mathbb{G}_1$ polynomial has degree at most $2q_I + 1$: initial $\mathbb{G}_1$ handles have degree at most one, and each issuance response adds the factor $K_{\mathsf{sid}}X$. This bound holds even when a later blinded message contains earlier issuance responses. Every $\mathbb{G}_2$ polynomial has degree at most two, since $\mathcal{A}$ only receives linear combinations of $[\![1]\!]_2$, $\mathsf{pk}_{\mathsf{iss}}$, and the commitments $[\![K_{\mathsf{sid}}X]\!]_2$. Therefore every $\mathbb{G}_T$ polynomial has degree at most $2q_I + 3$.

For any two distinct formal polynomials P, P' in the same group, the polynomial $P - P'$ is nonzero and has degree at most $2q_I + 3$ in the remaining hidden variables. Since X and the unrevealed K_{sid}'s are sampled from $\mathbb{Z}_p^\star$, and the H_t's are sampled from $\mathbb{Z}_p$, the Schwartz–Zippel lemma gives $\Pr[P^{\mathrm{ev}} = P'^{\mathrm{ev}}] \le \frac{2q_I+3}{p-1}$. A union bound over at most N_h^2 pairs of handles gives $\Pr[\mathsf{Coll}] \le \frac{(2q_I+3)N_h^2}{p-1}$. Condition on $\neg\mathsf{Coll}$ for the rest of the proof. Then equality of generic-group handles implies equality of their formal polynomials.

Let $\mathcal{X}$ be the set of executed sessions and $\mathcal{U}$ be the set of unexecuted sessions at the end of the game. After substituting $K_{\mathsf{sid}} = k_{\mathsf{sid}}$ for every $\mathsf{sid} \in \mathcal{X}$, we work over $\mathcal{R}^\star := \mathbb{Z}_p\left[X, \{H_t\}_{t\in[q_H]}, \{K_{\mathsf{sid}}\}_{\mathsf{sid}\in\mathcal{U}}\right]$.

The rank map. Define the $\mathbb{Z}_p$-linear map $\Phi : \mathcal{R}^\star \to \mathbb{Z}_p^{1+q_H}$ as follows. For any polynomial $P \in \mathcal{R}^\star$, write

$$P = X\left(b_0 + \sum_{t=1}^{q_H} b_t H_t\right) + P_\perp,$$

where $P_\perp$ is an $\mathbb{Z}_p$-linear combination of monomials outside $\{X, XH_1, \ldots, XH_{q_H}\}$. We set $\Phi(P) := (b_0, b_1, \ldots, b_{q_H})$. Thus, Φ reads off the X-linear, mask-free layer of a $\mathbb{G}_1$ polynomial.

Let

$$V := \mathrm{span}_{\mathbb{Z}_p}\left\{\Phi(\tilde{\sigma}^\star_{\mathsf{sid},i}) : \mathsf{sid} \in \mathcal{X},\ i \in [\ell_{\mathsf{sid}}]\right\},$$

where $\tilde{\sigma}^\star_{\mathsf{sid},i}$ denotes the issuance-response polynomial after the above partial substitution. The space V has one generator for each executed token position. Hence $\dim(V) \le \sum_{\mathsf{sid}\in\mathcal{X}} \ell_{\mathsf{sid}} \le N_{\mathsf{ex}}$.

Reachable $\mathbb{G}_1$ handles. We claim that every $\mathbb{G}_1$ handle available to $\mathcal{A}$, represented by a polynomial $A \in \mathcal{R}^\star$,

satisfies $\Phi(A) \in V$. This follows by induction over the symbolic execution. The initial $\mathbb{G}_1$ handle $[\![1]\!]_1$ and every random-oracle handle $[\![H_t]\!]_1$ have no factor of X, so their image under Φ is zero. Generic $\mathbb{G}_1$ operations preserve the claim by linearity. Pairing queries return only $\mathbb{G}_T$ handles, and Type-3 generic groups provide no map from $\mathbb{G}_T$ or $\mathbb{G}_2$ back to $\mathbb{G}_1$.

It remains to consider issuance responses. If $\mathsf{sid} \in \mathcal{X}$, then $\Phi(\tilde{\sigma}^{\star}_{\mathsf{sid},i}) \in V$ by definition. If $\mathsf{sid} \in \mathcal{U}$, then $\tilde{\sigma}^{\star}_{\mathsf{sid},i} = K_{\mathsf{sid}} X P^{\star}_{\mathsf{sid},i}$. Since K_{sid} was fresh when $\mathcal{A}$ submitted $P_{\mathsf{sid},i}$, it does not occur in $P^{\star}_{\mathsf{sid},i}$. Hence every monomial of $\tilde{\sigma}^{\star}_{\mathsf{sid},i}$ contains the unrevealed mask K_{sid}. Therefore no monomial is equal to X or XH_t, and $\Phi(\tilde{\sigma}^{\star}_{\mathsf{sid},i}) = 0$. This proves the claim.

Accepted tokens. Suppose $\mathcal{A}$ produces an accepted token signature α on an ephemeral public key $\mathsf{pk}_{\mathrm{eph}}$, and let $\mathsf{H}(\mathsf{pk}_{\mathrm{eph}}) = [\![H_c]\!]_1$. Let $A \in \mathcal{R}^{\star}$ be the $\mathbb{G}_1$ polynomial represented by α. The Spend and Refund procedures accept only if $\alpha \circ [\![1]\!]_2 = \mathsf{H}(\mathsf{pk}_{\mathrm{eph}}) \circ \mathsf{pk}_{\mathrm{iss}}$. The left-hand side has $\mathbb{G}_T$ polynomial A, while the right-hand side has $\mathbb{G}_T$ polynomial XH_c. Since we condition on $\neg\mathsf{Coll}$, we have the formal equality $A = XH_c$. This also covers the case where $\mathsf{H}(\mathsf{pk}_{\mathrm{eph}})$ is first queried internally by the verification procedure.

Therefore $\Phi(A) = \mathbf{e}_c$, where $\mathbf{e}_c$ is the standard basis vector whose H_c-coordinate is 1 and whose other coordinates are zero. By the reachability claim above, $\mathbf{e}_c \in V$.

Counting. Assume the adversary wins the issuer-fairness game, and let $M := N_{\mathsf{spd}} + N_{\mathsf{ref}}$. Then $M > N_{\mathsf{ex}}$. Each successful spend contributes one accepted token signature, and each successful refund contributes one accepted token signature for each refunded token. Since these signatures are on distinct messages, the game accepts M signatures on M distinct ephemeral public keys.

Let the corresponding random-oracle variables be $H_{c(1)}, \ldots, H_{c(M)}$. Since the ephemeral public keys are distinct, these variables are distinct. Thus $\mathbf{e}_{c(1)}, \ldots, \mathbf{e}_{c(M)}$ are linearly independent in $\mathbb{Z}_p^{1+q_H}$. By the previous paragraph, all these vectors lie in V. Hence $M \le \dim(V) \le N_{\mathsf{ex}}$, which contradicts $M > N_{\mathsf{ex}}$.

Thus, conditioned on $\neg\mathsf{Coll}$, no adversary wins the game. Therefore

$$\Pr[\mathcal{G}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}_{\mathsf{issuer\text{-}fair}}(1^{\lambda}) \Rightarrow 1] \le \Pr[\mathsf{Coll}] \le \frac{(2q_I + 3)N_h^2}{p - 1}. \quad \square$$

7. Implementation and Evaluation

We implemented our BAT schemes (Fig. 7 and Fig. 8) in Rust, and evaluated the performance of each phase.[1] We report execution time and communication cost, and break them down into client-side, issuer-side, and on-chain (environment-side) costs. In addition, we implement the on-chain environments of both protocols as Ethereum smart contracts, deploy them on the Sepolia testnet, and report measured gas costs in Section 7.3.

We henceforth refer to our protocol in Fig. 7 as Protocol A, and the protocol that facilitates non-interactive refunds in Fig. 8 as Protocol B.

For Protocol A, we use the BLS12-381 curve [75], [76] for issuer signatures and Schnorr signatures [77] over the edwards25519 curve [78] for clients' signatures on the ephemeral keys. For Protocol B, we use the BN254 curve [79] for both issuer and client signatures. We use the BN254 curve as it supports pairing-friendly curve cycles that enable more efficient zero-knowledge proofs.

We benchmark both protocols for batch sizes $\ell \in \{10, 50, 100\}$. We measure *single-threaded* computation costs (on-chain and off-chain) on an Apple M3 Max with 48 GB RAM, and we compute communication costs using serialized message payload sizes.

7.1. Implementation Details

In this section we describe additional implementation details and benchmarks results.

Protocol A. Recall from Section 5, in Protocol A, refunds require revealing each unspent token individually. We also implement an optimization in which, during refunds, we aggregate all BLS signatures and verify all refunded tokens using only one multi-pairing of length two.

Protocol B. We leverage a cycle of curves (BN254/Grumpkin[2]) to ensure that all elliptic curve operations are performed within the zero-knowledge (ZK) circuit over native fields, avoiding costly non-native emulations. Also, we use $\mathbb{G}_1$ and $\mathbb{G}_2$ elements of BN254 for ephemeral public keys and issuer public keys, respectively.

Recursive folding. For efficiency, we use Nova [81] as the backend for our succinct non-interactive argument of knowledge (SNARK). We implement the Nova folding engine over the Grumpkin curve. Because Nova proofs are relatively large, we recursively verify the Nova verifier using the Spartan SNARK protocol [82]. We use Spartan instead of Groth16 [83] because Groth16 SNARK protocol requires a bilinear pairing, and Grumpkin is not pairing-friendly.

Since a production-ready Spartan-on-Grumpkin SNARK protocol is not yet available, we adopt a hybrid benchmarking methodology. We measure the Nova folding time directly on Grumpkin using our implementation and use the official Spartan implementation on Ristretto255 [84] as a proxy for the final compression step. Ristretto255 and Grumpkin are both ≈254-bit prime-order groups with comparable arithmetic complexity, making this a conservative proxy.

For a batch of 64 tokens, folding requires 4.40 seconds, while Spartan compression of an equivalently sized circuit requires 3.70 seconds, yielding an end-to-end client latency of approximately 8.1 seconds.

1. Our implementation, including both the native and the smart-contract versions, is available at https://anonymous.4open.science/r/bat-implementation-6633/.

2. The Grumpkin curve [80] is a Weierstrass elliptic curve designed for SNARK-efficient group operations as a cycle curve to the BN254 curve.

	$\ell = 10$		$\ell = 50$		$\ell = 100$	
Component	**Time (ms)**	**Size (B)**	**Time (ms)**	**Size (B)**	**Time (ms)**	**Size (B)**
Client (off-chain issuance)	3.27 + 7.86	512	16.37 + 31.40	2,432	32.69 + 60.72	4,832
Issuer (off-chain issuance)	3.80	608	16.44	2,528	32.23	4,928
Client (unblind & bind)	2.38	–	11.52	–	22.96	–
On-chain exchange	0.61	264	0.60	264	0.61	264
On-chain spend ($\ell = 1$)	1.68	144	1.68	144	1.70	144
On-chain refund	7.04	1,440	28.79	7,200	55.87	14,400

TABLE 1: *Protocol A: token-reveal refund using BLS12-381 and Ed25519. Time denotes local execution time of the corresponding component (client, issuer, or environment). For client off-chain issuance, time shows blinding + verification for the two protocol rounds. All times are for single-threaded execution. Size is serialized message size (bytes).*

	$\ell = 10$		$\ell = 50$		$\ell = 100$	
Component	**Time (ms)**	**Size (B)**	**Time (ms)**	**Size (B)**	**Time (ms)**	**Size (B)**
Client (off-chain issuance)	3,614.40 + 2.32	85,224	7,137.84 + 6.20	86,504	12,163.80 + 11.06	88,104
Issuer (off-chain issuance)	1.22	480	5.05	1,760	9.72	3,360
Client (unblind & bind)	0.91	–	2.78	–	5.15	–
On-chain exchange	0.19	296	0.19	296	0.19	296
On-chain spend ($\ell = 1$)	2.02	128	2.04	128	2.04	128
On-chain refund	1.79	32	8.86	32	17.74	32

TABLE 2: *Protocol B: seed-reveal refund using BN254. We measure the time and size as in Table 1.*

7.2. Microbenchmark Results

We report our benchmark results for Protocol A and Protocol B, in Table 1 and Table 2, respectively. One BAT token is 144 bytes long, takes 0.24 milliseconds to generate (unblinding and binding), and 1.68 milliseconds to verify. In comparison, proofs for Zcash orchard—the version of Zcash with transparent setup—are 7KB long, take 1.7 seconds to generate, and 15 milliseconds to verify [21]. We compare against Zcash as it is the closest baseline to BAT in terms of not having threshold or trusted issuer, but note that Zcash is a more general scheme offering confidentiality of transaction amounts and recipients (c.f. Section 1).

Protocol A vs. Protocol B Our evaluation reveals a clean trade-off. Protocol A minimizes client-side cost but incurs linear communication cost for refunds. Protocol B shifts cost to the client through zero-knowledge proving during issuance, enabling compact, pairing-free refunds with constant on-chain communication. As we illustrate in Table 2, ZK proof generation dominates the client-side cost in Protocol B by several orders of magnitude. Finally, the same operations in Protocol B are faster than in Protocol A, as they are performed over the more efficient BN254 curve compared to BLS12-381, except for spends which use Ed25519 signatures in Protocol A.

7.3. On-chain Gas Costs

To validate BAT's deployability at the application layer, we implement the on-chain environments of both protocols as Ethereum smart contracts, and report the receipt-level gas costs in Table 3. Both contracts are deployed with verified source code on the Sepolia testnet[3].

3. Protocol A at 0x15ce…c2bd and Protocol B at 0x8526…1205.

Operation	**Protocol A**	**Protocol B**
Contract deployment	2,197,602	1,953,404
Register (issuer)	96,194	184,412
Exchange: client message (any ℓ)	124,037	142,681
Exchange: issuer message (any ℓ)	71,394	168,909
$\rightarrow$ per token at $\ell = 100$	1,954	3,116
Spend	198,942	207,000
Refund, $\ell_{\text{ref}} = 1$	207,498	90,092
Refund, $\ell_{\text{ref}} = 10$	629,875	368,745
Refund, $\ell_{\text{ref}} = 100$	5,096,298	3,164,381
$\rightarrow$ per token at $\ell = 100$	≈51,000	≈31,000

TABLE 3: *Gas costs for* BAT *smart contracts measured on Ethereum. Protocol A uses the EIP-2537 BLS12-381 precompiles; Protocol B uses the EIP-196/197 BN254 precompiles. Both contracts are deployed on the Sepolia testnet.*

Protocol A uses the BLS12-381 precompiles introduced in the Pectra upgrade (EIP-2537) for pairing checks, subgroup checks, and the map-to-curve step; the RFC 9380 hash-to-curve construction, which EIP-2537 deliberately leaves to the application, is implemented in Solidity as well. For the client signature scheme DS we use ECDSA over secp256k1, Ethereum's native scheme, so ephemeral public keys are Ethereum addresses and signature verification uses the 3,000-gas `ecrecover` precompile; we verify that the contract's hash-to-curve agrees byte-for-byte with our Rust implementation. The two on-chain messages of Π-Execute map to one transaction each; the client's payment is attached with its message, and can be reclaimed after δ_{inc} blocks if the issuer fails to reveal the masking key, mirroring the fact that in the model a client whose issuer aborts is never debited.

Protocol B is instantiated over BN254: ephemeral keys are BN254 $\mathbb{G}_1$ points derived on-chain as $\mathsf{PRF}(\mathsf{seed}, i) \cdot [\![1]\!]_1$ via the 6,000-gas `ecMul` precompile, the client's token binding is a Schnorr signature over BN254, and the issuer's masked-key opening is checked with one pairing product.

Our benchmarks reveal that BAT spend costs about 200k gas—$1.5\times$ cheaper than the 301,233 gas of a Tornado Cash withdrawal, and $6.9\times$ cheaper than a full Tornado Cash deposit–withdraw cycle (1,389,587 gas) [85]. Compact issuance amortizes the exchange to about 1,954 gas per token at $\ell = 100$. At the time of writing, these gas costs translate to about $0.60 per spend and well under a cent per issued token on Ethereum, which is the most expensive among modern EVM chains by orders of magnitude. Protocol B's higher issuance cost is an EVM artifact of the different curves used in the two implementations.

8. Discussion for Practical Deployments

Token denomination. We described our BAT protocol assuming the anonymous tokens have a fixed denomination. This simplification forces users to obtain multiple tokens for high-value transactions. Practical deployments should support different denomination pools for efficiency. Supporting multiple denominations requires no changes to the protocol: the issuer registers a separate issuing key per denomination (e.g., powers of two), and each key runs an independent BAT instance with its own escrow, counters, nullifier set, and retirement deadline. A spend under the denomination-d key mints d units and a refund of ℓ_{ref} tokens mints $d \cdot \ell_{\text{ref}}$ units. Clients can pay an arbitrary value V in two ways:

Powers-of-two pools. With denominations $1, 2, 4, \ldots$, a client pays V exactly by spending at most $\lceil \log_2 V \rceil + 1$ tokens. We note that the different denomination pools don't share anonymity sets among each other.

Change amounts. Alternatively, with a single coarse denomination, the client overpays and the change is credited to an address named during the spend. For certain usecases such as MEV protection, change address is only revealed after transaction inclusion along with other transaction data. For privacy-sensitive applications such as on-ramps for privacy protocols, later spending of the change links it to the original spend; a client can restore unlinkability by using the change to purchase fresh BAT tokens. The two approaches can be combined to balance the tradeoffs of overpayment and anonymity set size.

Non-interactive refunds and unlinkability. In the alternate scheme we describe in Fig. 8, we reveal the random seed that is used to generate all the ephemeral public keys. This links even the spent tokens at the time of refund. While this is acceptable for applications like MEV mitigation and on-chain voting, where the sender is revealed after a deadline, it may be problematic for other applications, such as on-ramp to private payment systems. If the anonymity of spent tokens is essential, our first construction should be used.

DoS prevention by malicious issuer. In our proposed BAT construction, a malicious issuer could prevent clients from spending the anonymous tokens it issues by exhausting its on-chain spend budget before clients attempt to spend their tokens. To disincentivize such issuers, clients can locally monitor their behavior and switch to more reputable issuers. We note that this attack does not allow a malicious issuer to steal tokens from honest clients, since clients can always claim refunds for any unspent tokens.

Choosing the retirement deadline T_{ret} and dealing with nullifier growth. T_{ret} should be long enough that an issuer accumulates a large anonymity set over its lifetime; deployments on the order of months are reasonable for high-throughput chains. Crucially, unlike Zcash-style systems where the nullifier set grows monotonically and forever, BAT discards an issuer's entire nullifier state at retirement, limiting the growth of nullifier set. Each nullifier is a 32-byte ephemeral public key, so merely 32 MB of storage per issuer supports an anonymity set of 10^6 spent tokens. While alternatives that try to maintain only accumulators on-chain come with significant practical tradeoffs, BAT does not preclude future optimizations through such techniques.

Choosing the grace period T_{grace} for pausing issuance. $T_{\mathsf{grace}} \geq 2\delta_{\mathsf{inc}}$ suffices for client fairness (Lemma 6.2), so T_{grace} can be set to a small constant. For instance, on Ethereum, where δ_{inc} is on the order of minutes, T_{grace} on the order of tens of minutes suffices; this is negligible compared to an active window T_{ret} on the order of months.

Capital efficiency. An issuer supporting ℓ outstanding tokens must escrow at least 2ℓ units for the T_{ret} window, incurring a cost of capital commensurate with ℓ that it offsets by charging a per-issuance fee.

Acknowledgments

The authors would like to thank Michael Setrin for many helpful discussions related to this project. Ari Juels would like to acknowledge the generous support of IC3 partners and NSF CNS-2427390.

References

[1] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 127–140.

[2] F. Beres, I. A. Seres, A. A. Benczur, and M. Quintyne-Collins, " Blockchain is Watching You: Profiling and Deanonymizing Ethereum Users ," in *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2021, pp. 69–78. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/DAPPS52256.2021.00013

[3] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 463–477.

[4] A. Pertsev, R. Semenov, and R. Storm, "Tornado cash privacy solution version 1.4," 2019, accessed: 2026-02-01. [Online]. Available: https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf

[5] V. Buterin, J. Illum, M. Nadler, F. Schär, and A. Soleimani, "Blockchain privacy and regulatory compliance: Towards a practical equilibrium," *Blockchain: Research and Applications*, vol. 5, no. 1, p. 100176, 2024.

[6] F. Baldimtsi, K. K. Chalkias, V. Madathil, and A. Roy, "SoK: Privacy-preserving transactions in blockchains," Cryptology ePrint Archive, Paper 2024/1959, 2024.

[7] P. Garimidi, J. Bonneau, and L. Heimbach. (2025, Jul.) On the limits of encrypted mempools. A16z Crypto. [Online]. Available: https://a16zcrypto.com/posts/article/limits-encrypted-mempools/

[8] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 910–927.

[9] K. Babel, P. Daian, M. Kelkar, and A. Juels, "Clockwork finance: Automated analysis of economic security in smart contracts," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2499–2516.

[10] B. Weintraub, C. F. Torres, C. Nita-Rotaru, and R. State, "A flash (bot) in the pan: measuring maximal extractable value in private pools," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 458–471.

[11] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?" in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 198–214.

[12] A. Agarwal, R. Fernando, and B. Pinkas, "Efficiently-thresholdizable batched identity based encryption, with applications," in *Advances in Cryptology – CRYPTO 2025*, Y. Tauman Kalai and S. F. Kamara, Eds. Cham: Springer Nature Switzerland, 2025, pp. 69–100.

[13] J. Bormet, S. Faust, H. Othman, and Z. Qu, "{BEAT-MEV}: Epochless approach to batched threshold encryption for {MEV} prevention," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 3457–3476.

[14] D. Boneh, B. Bünz, K. Nayak, L. Rotem, and V. Shoup, "Context-dependent threshold decryption and its applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2025, pp. 506–538.

[15] A. Agarwal, K. Babel, S. Das, B. P. Gilkalaye, A. Mondal, B. Pinkas, P. Rindal, and A. Yadav, "Weighted batched threshold encryption with applications to mempool privacy," Cryptology ePrint Archive, Paper 2025/2115, 2025.

[16] A. R. Choudhuri, S. Garg, J. Piet, and G.-V. Policharla, "Mempool privacy via batched threshold encryption: Attacks and defenses," in *USENIX Security 2024*, 2024.

[17] S. Suegami, S. Ashizawa, and K. Shibano, "Constant-cost batched partial decryption in threshold encryption," *Cryptology ePrint Archive*, 2024.

[18] A. R. Choudhuri, S. Garg, G. V. Policharla, and M. Wang, "Practical mempool privacy via one-time setup batched threshold encryption," in *34th USENIX Security Symposium*, 2025.

[19] J. Bormet, A. R. Choudhuri, S. Faust, S. Garg, H. Othman, G.-V. Policharla, Z. Qu, and M. Wang, "Beast-mev: Batched threshold encryption with silent setup for mev prevention," *Cryptology ePrint Archive*, 2025.

[20] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO'82*, 1982, pp. 199–203.

[21] M. Campanelli, M. Hall-Andersen, and S. H. Kamp, "Curve trees: Practical and transparent {Zero-Knowledge} accumulators," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4391–4408.

[22] E. N. Tas, I. A. Seres, Y. Zhang, M. Melczer, M. Kelkar, J. Bonneau, and V. Nikolaenko, "Atomic and fair data exchange via blockchain," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3227–3241.

[23] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *International Workshop on Public Key Cryptography*. Springer, 2002, pp. 31–46.

[24] M. K. Reiter and K. P. Birman, "How to securely replicate services," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 986–1009, 1994.

[25] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.

[26] D. Boneh, E. Laufer, and E. N. Tas, "Batch decryption without epochs and its application to encrypted mempools," *Cryptology ePrint Archive*, 2025.

[27] Shutter Network. Shutter network introduces plan for first encrypted mempool on ethereum. Globe Newswire. [Online]. Available: https://www.globenewswire.com/news-release/2025/02/13/3025629/0/en/Shutter-Network-Introduces-Plan-for-First-Encrypted-Mempool-on-Ethereum.html

[28] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 61–70.

[29] T. Lloyd, D. O'Broin, and M. Harrigan, "Emergent outcomes of the vetoken model," in *2023 IEEE international conference on omni-layer intelligent systems (COINS)*. IEEE, 2023.

[30] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso, "{VoteAgain}: A scalable coercion-resistant voting system," in *29th USENIX security symposium (USENIX Security 20)*, 2020, pp. 1553–1570.

[31] Privacy and Scaling Explorations. (Referenced Jan. 2026) Introduction to MACI. [Online]. Available: https://maci.pse.dev/docs/introduction

[32] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *FC 2020*, 2020, pp. 423–443.

[33] Railgun. (2024) Railgun documentation: Community broadcasters. [Online]. Available: https://docs.railgun.org/wiki/learn/privacy-system/community-broadcasters

[34] Z. Wang, S. Chaliasos, K. Qin, L. Zhou, L. Gao, P. Berrang, B. Livshits, and A. Gervais, "On how zero-knowledge proof blockchain mixers improve, and worsen user privacy," in *Proceedings of the ACM Web Conference 2023 (WWW '23)*. ACM, 2023.

[35] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously," *PoPETs*, no. 3, pp. 164–180, 2018.

[36] Apple, "icloud private relay overview," https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf, 2021, accessed: 2024-04-29.

[37] T. Silde and M. Strand, "Anonymous tokens with public metadata and applications to private contact tracing," in *FC 2022*, 2022, pp. 179–199.

[38] J. Wilander, "Introducing private click measurement, pcm," https://webkit.org/blog/11529/introducing-private-click-measurement-pcm/, 2021, accessed: 2024-04-29.

[39] B. Kreuter, T. Lepoint, M. Orrù, and M. Raykova, "Anonymous tokens with private metadata bit," in *CRYPTO 2020, Part I*, 2020, pp. 308–336.

[40] F. Benhamouda, T. Lepoint, M. Orrù, and M. Raykova, "Publicly verifiable anonymous tokens with private metadata bit," Cryptology ePrint Archive, Report 2022/004, 2022.

[41] M. Chase, F. B. Durak, and S. Vaudenay, "Anonymous tokens with stronger metadata bit hiding from algebraic MACs," in *CRYPTO 2023, Part II*, 2023, pp. 418–449.

[42] M. Orrù, S. Tessaro, G. Zaverucha, and C. Zhu, "Oblivious issuance of proofs," in *CRYPTO 2024, Part IX*, 2024, pp. 254–287.

[43] F. Baldimtsi, L. Hanzlik, Q. Nguyen, and A. Yadav, "Non-interactive anonymous tokens with private metadata bit," Cryptology ePrint Archive, Report 2025/430, 2025.

[44] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *CRYPTO'88*, 1990, pp. 319–327.

[45] F. Baldimtsi, M. Chase, G. Fuchsbauer, and M. Kohlweiss, "Anonymous transferable E-cash," in *PKC 2015*, 2015, pp. 101–124.

[46] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," Bitcointalk Forum, 2013. [Online]. Available: https://bitcointalk.org/index.php?topic=279249.0

[47] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "CoinShuffle: Practical decentralized coin mixing for Bitcoin," in *ESORICS 2014, Part II*, 2014, pp. 345–364.

[48] S. Meiklejohn and C. Orlandi, "Privacy-enhancing overlays in Bitcoin," in *FC 2015 Workshops*, 2015, pp. 127–141.

[49] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain Bitcoin transactions," in *FC 2016 Workshops*, 2016, pp. 43–60.

[50] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub," in *NDSS 2017*. The Internet Society, 2017.

[51] S. Meiklejohn and R. Mercer, "Möbius: Trustless tumbling for transaction privacy," *PoPETs*, no. 2, pp. 105–121, 2018.

[52] I. A. Seres, D. A. Nagy, C. Buckland, and P. Burcsi, "MixEth: Efficient, trustless coin mixing service for Ethereum," in *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*, ser. OASIcs, vol. 71. Schloss Dagstuhl, 2019.

[53] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 104–121.

[54] Á. Ficsór, I. A. Seres, Y. Kogman, and L. Ontivero, "Wabisabi: Centrally Coordinated CoinJoins with Variable Amounts," *Cryptoeconomic Systems*, vol. 1, no. 2, oct 22 2021, https://cryptoeconomicsystems.pubpub.org/pub/ficsor-wabisabi-coordinated.

[55] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed E-cash from Bitcoin," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 397–411.

[56] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 459–474.

[57] N. van Saberhagen, "CryptoNote v 2.0," https://cryptonote.org/whitepaper.pdf, 2013, the foundational protocol for Monero.

[58] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *ASIACRYPT 2001*, 2001, pp. 552–565.

[59] B. E. Diamond, "Many-out-of-many proofs and applications to anonymous zether," in *2021 IEEE Symposium on Security and Privacy*, 2021, pp. 1800–1817.

[60] A. Tomescu, A. Bhat, B. Applebaum, I. Abraham, G. Gueta, B. Pinkas, and A. Yanai, "UTT: Decentralized ecash with accountable privacy," Cryptology ePrint Archive, Report 2022/452, 2022.

[61] N. Narula, W. Vasquez, and M. Virza, "zkLedger: Privacy-preserving auditing for distributed ledgers," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 65–80.

[62] K. Wüst, K. Kostiainen, N. Delius, and S. Capkun, "Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation," in *ACM CCS 2022*, 2022, pp. 2947–2960.

[63] A. Kiayias, M. Kohlweiss, and A. Sarencheh, "PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies," in *ACM CCS 2022*, 2022, pp. 1739–1752.

[64] A. Sarencheh, A. Kiayias, and M. Kohlweiss, "PARScoin: A privacy-preserving, auditable, and regulation-friendly stablecoin," in *CANS 2024, Part I*, 2024, pp. 289–313.

[65] A. Sarencheh, H. Khoshakhlagh, A. Kavousi, and A. Kiayias, "DART: Decentralized, anonymous, and regulation-friendly tokenization," Cryptology ePrint Archive, Report 2025/239, 2025.

[66] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *ASIACRYPT 2019, Part I*, 2019, pp. 649–678.

[67] M. Campanelli and M. Hall-Andersen, "Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions," in *ASIACCS 22*, 2022, pp. 652–666.

[68] V. Madathil and A. Scafuro, "PriFHEte: Achieving full-privacy in account-based cryptocurrencies is possible," Cryptology ePrint Archive, Report 2023/710, 2023.

[69] A. B. Hansen, J. B. Nielsen, and M. Simkin, "OCash: Fully anonymous payments between blockchain light clients," in *PKC 2025, Part V*, 2025, pp. 169–202.

[70] N. T. Courtois and R. Mercer, "Stealth address and key management techniques in blockchain systems," in *3rd International Conference on Information Systems Security and Privacy (ICISSP)*, 2017.

[71] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *ASIACRYPT 2001*, 2001, pp. 514–532.

[72] N. Gailly, K. Melissaris, and Y. Romailler, "tlock: Practical timelock encryption from threshold BLS," Cryptology ePrint Archive, Paper 2023/189.

[73] A. Agarwal, K. Babel, S. Das, and B. P. Gilkalaye, "Time-lock encrypted storage for blockchains," Cryptology ePrint Archive, Paper 2025/2048.

[74] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1997, pp. 256–266.

[75] P. S. L. M. Barreto, B. Lynn, and M. Scott, "Efficient implementation of pairing-based cryptosystems," *Journal of Cryptology*, no. 4, pp. 321–334, 2004.

[76] S. Bowe, "Bls12-381: New zk-snark elliptic curve construction," https://electriccoin.co/blog/new-snark-curve/, Mar 11 2017, electric Coin Company Blog.

[77] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, no. 3, pp. 161–174, 1991.

[78] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," in *CHES 2011*, 2011, pp. 124–142.

[79] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *SAC 2005*, 2006, pp. 319–331.

[80] A. Protocol, "Primitives - Aztec Connect: Grumpkin - A curve on top of BN-254 for SNARK efficient group operations," https://tinyurl.com/aztec-grumpkin, 2024, elliptic curve used for native recursion in Aztec zk-SNARK systems.

[81] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *CRYPTO 2022, Part IV*, 2022, pp. 359–388.

[82] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in *CRYPTO 2020, Part III*, 2020, pp. 704–737.

[83] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, 2016, pp. 305–326.

[84] M. Hamburg, "Decaf: Eliminating cofactors through point compression," in *CRYPTO 2015, Part I*, 2015, pp. 705–723.

[85] Tornado Cash. (2019) tornado-core: Reference implementation. [Online]. Available: https://github.com/tornadocash/tornado-core

Requirements: Digital signature DS, pairing groups $\mathcal{G} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, e)$, hash $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$, commitment Com, PRF PRF, and NIZK NIZK for $\mathcal{R}_{\text{seed}}$, where $\mathcal{R}_{\text{seed}} = \{((\mathsf{com}_{\text{seed}}, \boldsymbol{X}, \ell); (\text{seed}, \mathsf{d}_{\text{seed}}, r)) : \mathsf{Com.Verify}(\mathsf{com}_{\text{seed}}, \text{seed}, \mathsf{d}_{\text{seed}}) = 1 \wedge \forall i \in [\ell] : (\star, \mathsf{pk}_{\text{eph},i}) = \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}}; \mathsf{PRF}(\text{seed}, i)) \wedge x_i = \mathsf{H}(\mathsf{pk}_{\text{eph},i})^r\}$

Setup(1^λ)

$\mathsf{pp}_{\mathsf{DS}} \leftarrow \mathsf{DS.Setup}(1^\lambda)$
$\mathsf{pp}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda)$
$\mathsf{pp}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}_{\text{seed}})$
return $\mathsf{pp} := (\mathcal{G}, \mathsf{pp}_{\mathsf{DS}}, \mathsf{pp}_{\mathsf{Com}}, \mathsf{pp}_{\mathsf{NIZK}}, T_{\text{ret}}, T_{\text{grace}})$

IssKGen, Register, Retire

Same as Figure 7

Π-MIssue$(\mathsf{pk}_{\text{iss}}, \ell, \mathsf{sk}_{\text{iss}}, \mathsf{st}_{\mathcal{I}})$

$\mathcal{C}$: sid $\leftarrow\$ \{0,1\}^\lambda$; seed $\leftarrow\$ \{0,1\}^\lambda$
  $(\mathsf{com}_{\text{seed}}, \mathsf{d}_{\text{seed}}) \leftarrow \mathsf{Com.Commit}(\text{seed})$
  $\{(\mathsf{sk}_{\text{eph},i}, \mathsf{pk}_{\text{eph},i}) := \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}}; \mathsf{PRF}(\text{seed}, i))\}_{i\in[\ell]}$
  $r \leftarrow\$ \mathbb{Z}_p$; $\boldsymbol{X} := \{r \cdot H(\mathsf{pk}_{\text{eph},i})\}_{i\in[\ell]}$
  $\pi \leftarrow \mathsf{NIZK.Prove}(\mathcal{R}_{\text{seed}}, (\mathsf{com}_{\text{seed}}, \boldsymbol{X}, \ell), (\text{seed}, \mathsf{d}_{\text{seed}}, r))$
  send $(\text{sid}, \mathsf{com}_{\text{seed}}, \boldsymbol{X}, \ell, \pi)$ to $\mathcal{I}$
$\mathcal{I}$: **require** sid $\notin \mathsf{st}_{\mathcal{I}}$.sessions
  require $\mathsf{NIZK.Verify}(\mathcal{R}_{\text{seed}}, (\mathsf{com}_{\text{seed}}, \boldsymbol{X}, \ell), \pi) = 1$
  $k \leftarrow\$ \mathbb{Z}_p$; $\mathsf{com}_k \leftarrow k \cdot \mathsf{pk}_{\text{iss}}$
  $\mathsf{st}_{\mathcal{I}}$.sessions[sid] $:= (\mathsf{com}_k, \mathsf{com}_{\text{seed}}, \ell, k)$
  $\{\tilde{\sigma}_i := (k \cdot \mathsf{sk}_{\text{iss}}) \cdot X_i\}_{i\in[\ell]}$
  send $(\{\tilde{\sigma}_i\}_{i\in[\ell]}, \mathsf{com}_k)$ to $\mathcal{C}$
$\mathcal{C}$: **require** $\forall i \in [\ell] : \tilde{\sigma}_i \circ [\![1]\!]_2 = X_i \circ \mathsf{com}_k$
  $(\mathsf{sk}_{\text{ref}}, \mathsf{pk}_{\text{ref}}) \leftarrow \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}})$
  $w := (r, \text{seed}, \mathsf{d}_{\text{seed}})$
  $\mathsf{st}_{\mathcal{C}} := \left(\begin{array}{l} \{\tilde{\sigma}_i, \mathsf{sk}_{\text{eph},i}, \mathsf{pk}_{\text{eph},i}\}_{i\in[\ell]}, \\ \mathsf{sk}_{\text{ref}}, \mathsf{pk}_{\text{ref}}, \mathsf{com}_k, \mathsf{com}_{\text{seed}}, w \end{array} \right)$
  return $(\text{sid}, \mathsf{st}_{\mathcal{C}})$

TokGen, Spend

Same as Figure 7

Π-Execute$^{\mathcal{E}}(\text{sid}, \ell, \mathsf{st}_{\mathcal{I}}, \mathsf{st}_{\mathcal{C}})$

$\mathcal{C}$: *// Represented by* pk_{c}
  parse $\mathsf{st}_{\mathcal{C}}$ as $(\star, \star, \mathsf{pk}_{\text{ref}}, \mathsf{com}_k, \mathsf{com}_{\text{seed}}, \star)$
  send $(\text{sid}, \mathsf{pk}_{\text{ref}}, \mathsf{com}_k, \mathsf{com}_{\text{seed}})$ to $\mathcal{E}$
$\mathcal{I}$: *// Represented by* pk_{iss}
  require $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\text{c}}] \geq \ell$
  require $\mathsf{st}_{\mathcal{I}}.\mathsf{sessions}[\text{sid}] = (\mathsf{com}_k, \mathsf{com}_{\text{seed}}, \ell, \star)$
  require $2(\mathsf{st}_{\mathcal{I}}.\mathsf{Niss} + \ell) \leq \ell_{\text{iss}}$
  $\mathsf{st}_{\mathcal{I}}.\mathsf{Niss} := \mathsf{st}_{\mathcal{I}}.\mathsf{Niss} + \ell$ *// $\mathsf{st}_{\mathcal{I}}$.Niss initialized to 0*
  parse $\mathsf{st}_{\mathcal{I}}.\mathsf{sessions}[\text{sid}]$ as $(\star, \star, \star, k)$
  send (sid, k) to $\mathcal{E}$
$\mathcal{E}$: **require** $\mathsf{pk}_{\text{iss}} \notin \mathcal{E}.\mathsf{retired}$
  require $\mathcal{E}.\mathsf{currentHeight} + T_{\text{grace}} \leq \mathcal{E}.\mathsf{retireBlock}[\mathsf{pk}_{\text{iss}}]$
  require $k \cdot \mathsf{pk}_{\text{iss}} = \mathsf{com}_k$
  require $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}] + \ell \leq \frac{\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\text{iss}}]}{2}$
  $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}] := \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}] + \ell$
  $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\text{c}}] := \mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\text{c}}] - \ell$ *// Burn ℓ ETH*
  $\mathcal{E}.\mathsf{sessions}[\text{sid}] := (\mathsf{pk}_{\text{ref}}, \mathsf{com}_k, \mathsf{com}_{\text{seed}}, \ell)$
$\mathcal{C}$: *// Use k to unmask pre-tokens*
  parse $\mathsf{st}_{\mathcal{C}}$ as $(\{\tilde{\sigma}_i, \mathsf{sk}_{\text{eph},i}, \mathsf{pk}_{\text{eph},i}\}_{i\in[\ell]}, \star, \star, \star, \star, w)$
  parse w as $(r, \star, \star)$
  return $\{\sigma_i := ((r_i \cdot k)^{-1} \cdot \tilde{\sigma}_i, \mathsf{sk}_{\text{eph},i}, \mathsf{pk}_{\text{eph},i})\}_{i\in[\ell]}$

Refund$^{\mathcal{E}}(\text{sid}, \ell_{\text{ref}}, \mathsf{st}_{\mathcal{C}})$

$\mathcal{C}$: **parse** $\mathsf{st}_{\mathcal{C}}$ as $(\star, \mathsf{sk}_{\text{ref}}, \star, \star, \star, w)$
  parse w as $(\star, \text{seed}, \mathsf{d}_{\text{seed}})$
  $\gamma_{\text{ref}} \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\text{ref}}, (\text{sid}, \ell_{\text{ref}}, \text{seed}, \mathsf{d}_{\text{seed}}))$
  send $(\text{sid}, \ell_{\text{ref}}, \text{seed}, \mathsf{d}_{\text{seed}}, \gamma_{\text{ref}})$ to $\mathcal{E}$
$\mathcal{E}$: **require** $\mathsf{DS.Verify}(\mathsf{pk}_{\text{ref}}, (\text{sid}, \ell_{\text{ref}}, \text{seed}, \mathsf{d}_{\text{seed}}), \gamma_{\text{ref}}) = 1$
  require sid $\in \mathcal{E}.\mathsf{sessions}$ and sid $\notin \mathcal{E}.\mathsf{refundset}$
  parse $\mathcal{E}.\mathsf{sessions}[\text{sid}]$ as $(\mathsf{pk}_{\text{ref}}, \mathsf{com}_k, \mathsf{com}_{\text{seed}}, \ell)$
  require $\ell_{\text{ref}} \leq \ell$ and $\mathsf{pk}_{\text{iss}} \notin \mathcal{E}.\mathsf{retired}$
  require $\mathsf{Com.Verify}(\mathsf{com}_{\text{seed}}, \text{seed}, \mathsf{d}_{\text{seed}}) = 1$
  for $i \in (\ell - \ell_{\text{ref}}, \ell]$:
    $(\star, \mathsf{pk}_{\text{eph},i}) \leftarrow \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}}; \mathsf{PRF}(\text{seed}, i))$
  for $i \in (\ell - \ell_{\text{ref}}, \ell]$: **require** $\mathsf{pk}_{\text{eph},i} \notin \mathsf{Nullifiers}$
  $\mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\text{ref}}] := \mathcal{E}.\mathsf{bal}[\mathsf{pk}_{\text{ref}}] + \ell_{\text{ref}}$ *// Mint ℓ_{ref} ETH*
  $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\text{iss}}] := \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\text{iss}}] + \ell_{\text{ref}}$
  $\mathcal{E}.\mathsf{refundset} := \mathcal{E}.\mathsf{refundset} \cup \{\text{sid}\}$
  $\mathcal{E}.\mathsf{Nullifiers} := \mathcal{E}.\mathsf{Nullifiers} \cup \{\mathsf{pk}_{\text{eph},i} | i \in (\ell - \ell_{\text{ref}}, \ell]\}$

Figure 8: BAT *protocol with compact issuance and refunds based on revealing the seed.* $\mathcal{C}$ *: denotes actions by the client,* $\mathcal{I}$ *: by the issuer, and* $\mathcal{E}$ *: by the environment* $\mathcal{E}$*. Additional parts compared to Figure 7 are marked in blue.*