Cryptocurrency-Backed Trustless Anonymous Tokens and Their Applications

Amit Agarwal[1], Kushal Babel[1], Sourav Das[1], Ari Juels[2,3], Peter Rindal[1], Aayush Yadav[4]

[1]*Category Labs* [2]*Cornell Tech* [3]*IC3* [4]*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 $O(1)$ 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 assuming hardness of the one-more computational Diffie-Hellman assumption in a bilinear pairing group in the 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 more than 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 [13, 46, 54]. This limitation has motivated the development of smart-contract-based anonymity tools, such as Tornado Cash [57] and Privacy Pools [26], 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 transactions, such as fee payments. The need for private fee payment in particular is an acknowledged bottleneck in the construction of private transaction systems [6, 40].

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 costs) 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) [5, 35, 60, 74], 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 [2, 3, 17, 22, 23, 33, 34, 70] aim to hide transaction data to mitigate MEV attacks, they do not address the problem of sender metadata. Sender metadata can serve at the time of transaction inclusion to verify payment ability and thus prevent DoS attacks, but such metadata can itself facilitate MEV attacks—despite transaction encryption. 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 un-

derlying currency of the blockchain, a property BAT achieves.

We instantiate BAT as a form of classic e-cash system [30], 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 across blockchain layers. 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 available for redemption 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 selective security of our BAT scheme, assuming the hardness of the one-more computational Diffie-Hellman assumption in bilinear pairing groups in the random oracle model 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 published benchmarks for Zcash Orchard as a baseline [11], we show that BAT tokens are more than $50\times$ smaller in size, approximately $9\times$ faster to verify, and over $7{,}000\times$ 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 $[n] := [1,n]$, $x \leftarrow\!\$\, S$ 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 $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p, with a (non-degenerate) bilinear map or pairing $\circ : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The groups $(\mathbb{G}_1, \mathbb{G}_2)$ are referred to as the source groups, while $\mathbb{G}_T$ 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.

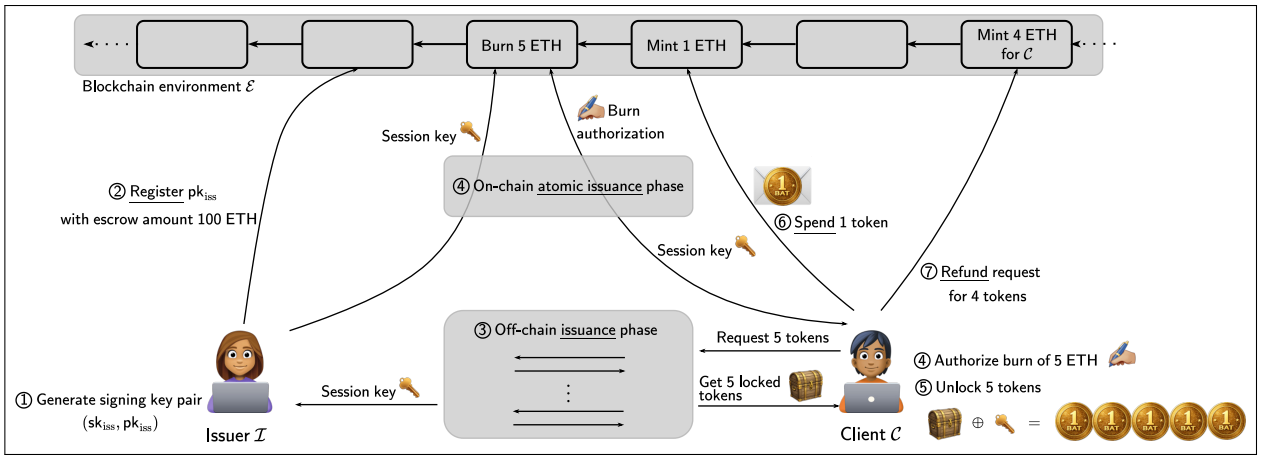

Figure 1: Overview of our blockchain anonymous token (BAT) architecture involving three entities - client, issuer, blockchain environment - and four main phases including registration, issuance, spend and refund. The circled numbers denote the sequence in which various steps are executed. The grey box 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.

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.

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 the issuer must post $O(\ell)$ communication on-chain, and $\mathcal{E}$ must validate all posted messages.

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 [71]. 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: our 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 cryptocurrency.

Registration phase. In this phase, any party that holds a private-public key pair $(\mathsf{sk}_{\text{iss}}, \mathsf{pk}_{\text{iss}}) \in \mathbb{Z}_p \times \mathbb{G}_1$ 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 [15] where for each $i \in [\ell]$, the i-th blind signatures is defined as $\tilde{\sigma}_i := \mathsf{sk}_{\text{iss}} \cdot M_i \in \mathbb{G}_2$, for some client chosen input $M_i \in \mathbb{G}_2$.

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. It is easy to see that 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 the sk_{eph}. The client then posts the tuple $\rho := (\sigma, \mathsf{pk}_{\text{eph}}, \gamma, \mathsf{ad})$ 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 of the token authorizes this specific spend 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 Section 6 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 use with the associated data.

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 $O(\ell_r)$ on-chain communication and computation. In Section 5, we present an optimized protocol that reduces on-chain communication to $O(1)$ and enables *automated* refunds without requiring 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 Section 6, this mechanism suffices to ensure protocol 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) [5, 35, 60, 74] 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 [18, 27, 61, 68]. 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., sandwiching of target senders' transactions.

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 [45]. In fact, bribery is already a practical issue for DAOs [49].

Coercion-resistant voting systems, e.g., [50, 58], require use of encrypted ballots. As in the case of MEV, however, preventing DoS attacks requires voters to pay transaction 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. Smart-contract systems such as Zether [25] 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.

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 enable users to prove possession of some form of authorization without revealing their identity or allowing linkability. This unlinkability is an essential in applications such as private web browsing [4, 36], private contact tracing [69], fraud detection [69], and private click measurement [75]. These applications have inspired a rich body of research on anonymous tokens and blind signatures, with different trust, efficiency, and security tradeoffs [8, 12, 29, 36, 48, 56, 69].

Anonymous tokens were originally devised as the basis for anonymous electronic cash (e-cash) systems [7, 32]. 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 itself.

Coin mixing and tumbling services. Cryptocurrency mixers for Bitcoin, such as CoinJoin [51], CoinShuffle [53, 63] and TumbleBit [43, 44] aim to provide transaction unlinkability by shuffling funds among multiple participants. Tornado Cash [57] and Möbius [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 [63], expensive cryptographic primitives such as zk-SNARKs in Tornado Cash [57] or ring signatures in Möbius [52], which often limits anonymity sets; or incur high latency due to epoch-based designs [43, 44]. 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.

The use of e-cash-inspired anonymous tokens for mixing has been explored in CoinJoin [51] and in the 'blindly signed contracts' of Heilman, et al. [44]. 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 [21], later improvements have had to rely on complex multi-party systems [38, 63]. In comparison, [44] 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 internal 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) [11] rely on zk-SNARKs to provide strong anonymity, but incur high computational costs or require a trusted setup. Monero [73] avoids trusted setup by using ring signatures [62], 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 plat-

forms. In the account-based model, Zether [25] 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 [37] 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. [72] 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.

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.

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 in interfaces to indicate that it is 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.

- $\mathsf{IssKGen}(\mathsf{pp}) \to (\mathsf{sk}_{\mathrm{iss}}, \mathsf{pk}_{\mathrm{iss}})$. An issuer uses the IssKGen algorithm to sample a secret issuing key $\mathsf{sk}_{\mathrm{iss}}$ and the corresponding public verification key $\mathsf{pk}_{\mathrm{iss}}$.

- $\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}, \ell_{\mathrm{iss}}) \to 0/1$. Any party uses the registration interface Register with its public key $\mathsf{pk}_{\mathrm{iss}}$ 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.

- $\mathsf{Retire}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}}) \to 0/1$. Any existing issuer uses the retire interface with its public key $\mathsf{pk}_{\mathrm{iss}}$. Upon successful execution, $\mathcal{E}$ returns $\ell'_{\mathrm{iss}} \in \mathbb{N}$ ETH from the issuer's escrow account to the issuer and outputs 1. Otherwise, it outputs 0.

- $\Pi\text{-}\mathsf{MIssue}(\mathsf{pk}_{\mathrm{iss}}, \ell, \mathsf{sk}_{\mathrm{iss}}, \mathsf{st}_I) \to (\mathsf{sid}, \mathsf{st}_I, \mathsf{st}_C)$. The masked issuance protocol is a two-party protocol between a client and an issuer with public inputs being the issuer's public key $\mathsf{pk}_{\mathrm{iss}}$, the number of requested tokens ℓ, and the issuer's private input being its secret issuance key $\mathsf{sk}_{\mathrm{iss}}$, and its internal state st_I. The protocol outputs a public session identifier sid. Additionally, it outputs the modified issuer's internal state and privately outputs its internal state st_C to the client.

- $\Pi\text{-}\mathsf{Execute}^{\mathcal{E}}(\mathsf{sid}, \ell, \mathsf{st}_I, \mathsf{st}_C) \to (\mathsf{st}_I, \{\sigma_j\}_{j\in[\ell]})$. Π-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 st_C, and the issuer's private internal state st_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 internal state st_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.

- $\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid}, \{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}]}, \mathsf{st}_C) \to 0/1$. Any client with its internal state st_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 now 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 $\mathcal{G}_{\mathsf{cor}}^{\mathsf{BAT},\mathcal{E}}$ 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".

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

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

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

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

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

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 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 $\mathcal{A}$,

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

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 $\{\mathsf{ad}_j\}_{j\in[\ell]} \in \{0,1\}^*$,

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

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

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

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

Oracle $\mathcal{O}$-MIssue(ℓ):

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

Oracle $\mathcal{O}$-Execute(sid,ℓ):

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

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$(\mathsf{sid},\{\sigma_j\}_{j\in[\ell_{\mathrm{ref}}],\mathsf{st}_C})$:

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

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

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 $\mathsf{Register}^{\mathcal{E}}(\mathsf{pk}_{\mathrm{iss}},\ell_{\mathrm{iss}}) = 1$
$(\mathsf{sid},\star,\mathsf{st}_C) \leftarrow \Pi\text{-MIssue}(\mathsf{pk}_{\mathrm{iss}},\ell,\star)$
$(\star,\{\sigma_j\}_{j\in[\ell]}) \leftarrow \Pi\text{-Execute}^{\mathcal{E}}(\mathsf{sid},\ell,\star,\mathsf{st}_C)$
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 $\mathsf{Refund}^{\mathcal{E}}(\mathsf{sid},\{\sigma_k\}_{k\in[j,\ell]},\mathsf{st}_C) = 0$
    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.

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

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

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

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.

sessions of issuance protocols and is later shown finalized tokens in a random order. It must then guess which issuance 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. From the adversary's perspective, 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,

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

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

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

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

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

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

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

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

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.

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 $\mathcal{G} =$

Requirements: Digital signature DS, pairing groups $\mathcal{G} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, \circ) \leftarrow \mathsf{GGen}(1^\lambda)$ 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** $\mathsf{pp} := (\mathcal{G}, \mathsf{pp}_{\mathsf{DS}})$

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

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 : 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 : $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}] := 0$; $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}] := 0$

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

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

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

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

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

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

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

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

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

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

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

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

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

$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, \circ) \leftarrow \mathsf{GGen}(1^\lambda)$, and public parameters $\mathsf{pp}_{\mathsf{DS}}$ of a digital signature scheme DS. We also use a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$ modeled as a random oracle.

Our protocol uses the blind BLS signature as anonymous tokens [19]. Thus the private issuing key of an issuer is the BLS secret key $\mathsf{sk}_{\mathsf{iss}} \leftarrow\!\$\, \mathbb{Z}_p$ and the associated public key is $\mathsf{pk}_{\mathsf{iss}} := [\![\mathsf{sk}_{\mathsf{iss}}]\!]_2$. Clients generate a key pair $(\mathsf{sk}_{\mathsf{ref}}, \mathsf{pk}_{\mathsf{ref}}) \leftarrow \mathsf{DS.KGen}(\mathsf{pp}_{\mathsf{DS}})$ to authorize and receive refunds.

Issuer registration. Any issuer I registers its public issuing key $\mathsf{pk}_{\mathsf{iss}}$ with $\mathcal{E}$ using the $\mathsf{Register}^{\mathcal{E}}$ interface by depositing ℓ units of ETH as collateral. The registration is successful if I has sufficient balance and has not already registered. Upon successful registration, $\mathcal{E}$ locks ℓ_{iss} ETH from the I's balance into an escrow account $\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}]$ and initializes counters: $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}] := 0$ to track the number of tokens issued by I, $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] := 0$ to track tracks of spent tokens, and $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}] := 0$ to track the number of refunded tokens. I also initializes a local variable $\mathsf{st}_I.\mathsf{Niss} := 0$ to track the number of anonymous token it has issued so far.

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

In the first move, $\mathcal{C}$ generates ℓ ephemeral key pairs $\{(\mathsf{sk}_{\mathsf{eph},i}, \mathsf{pk}_{\mathsf{eph},i})\}_{i\in[\ell]}$ 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 $\boldsymbol{X} = \{X_i = r_i \cdot H(\mathsf{pk}_{\mathsf{eph},i})\}_{i\in[\ell]}$, and sends them to I along with a random session identifier sid.

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

On-chain atomic issuance. The Π-Execute protocol coordinates the masking key revelation and payment for it via the blockchain $\mathcal{E}$. The client $\mathcal{C}$ submits $(\mathsf{sid}, \mathsf{pk}_{\mathsf{ref}}, \mathsf{com}_k, \ell)$ to $\mathcal{E}$, where $\mathsf{pk}_{\mathsf{ref}}$ is the refund public key, and escrows the payment for ℓ tokens. In response, I reveals the masking key k to $\mathcal{E}$, while ensuring that it does not issue more than $\ell_{\mathsf{iss}}/2$ tokens.

Upon receiving k, $\mathcal{E}$ checks: (1) k matches the commitment, i.e., $k \cdot \mathsf{pk}_{\mathsf{iss}} = \mathsf{com}_k$, (2) the issuer has not retired, and (3) the issuer has sufficient collateral to back this issuance via $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}] + \ell \le \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\mathsf{iss}}]/2$. This last 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 $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}]$ by ℓ, burns ℓ coins from the $\mathcal{C}$'s balance, and stores $(\mathsf{pk}_{\mathsf{ref}}, \mathsf{com}_k, \ell)$ for future potential 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 $\sigma_i := (\alpha_i, \mathsf{pk}_{\mathsf{eph},i}, \mathsf{sk}_{\mathsf{eph},i})$ as the i-th pre-token.

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

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

Upon successful spend, $\mathcal{E}$ adds $\mathsf{pk}_{\mathsf{eph}}$ to the nullifier set $\mathcal{E}.\mathsf{Nullifiers}$, increments $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}]$, 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 $\mathsf{sk}_{\mathsf{ref}}$. Let $\gamma_{\mathsf{ref}} := \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{ref}}, (\mathsf{sid}, \{\alpha_i, \mathsf{pk}_{\mathsf{eph},i}\}_{i\in[\ell_{\mathsf{ref}}]}))$ be the digital signature. $\mathcal{C}$ submits the refund claim posting $(\mathsf{sid}, \{\alpha_i, \mathsf{pk}_{\mathsf{eph},i}\}_{i\in[\ell_{\mathsf{ref}}]}, \gamma_{\mathsf{ref}})$ to $\mathcal{E}$.

Upon receiving the refund claim, $\mathcal{E}$ validates it by checking: (i) the session sid has not been refunded before, (ii) $\ell_{\mathsf{ref}} \le \ell$ where ℓ is the batch size from session sid; (iii) for each $i \in [\ell_{\mathsf{ref}}]$, α_i is a valid BLS signature, i.e., $\alpha_i \circ [\![1]\!]_2 = H(\mathsf{pk}_{\mathsf{eph},i}) \circ \mathsf{pk}_{\mathsf{iss}}$, and (iv) None of the ephemeral keys has been spent. If all checks pass, $\mathcal{E}$ mints ℓ_{ref} ETH coins to $\mathsf{pk}_{\mathsf{ref}}$, adds all ephemeral keys to the nullifier set, increments $\mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}]$ 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 $\alpha = \sum_{i=1}^{\ell_{\mathsf{ref}}} \alpha_i$ 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 I with public key $\mathsf{pk}_{\mathsf{iss}}$ submits a retirement request to recover its collateral, $\mathcal{E}$ computes $\Delta = \max(0, \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\mathsf{iss}}] + \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\mathsf{iss}}] - \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\mathsf{iss}}])$, which captures any deficit due to 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 I, burns Δ units of ETH, and marks $\mathsf{pk}_{\mathsf{iss}}$ as retired.

Towards non-interactive refunds. In Fig. 9, 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 $O(1)$ 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 I proving that the blinded values correspond to ephemeral keys correctly derived using a PRF on the seed. 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 $O(\ell_{\text{ref}})$ 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 [1, 39] 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 I escrows sufficient funds during registration. Recall that each masked pre-token in our scheme is of the form $\tilde{\sigma}_i = (k \cdot \mathsf{sk}_{\text{iss}}) \cdot X_i$ and $\mathsf{com}_k = k \cdot \mathsf{pk}_{\text{iss}}$. Thus, we get

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

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 $\alpha_i := (r \cdot k)^{-1} \cdot \tilde{\sigma}_i = \mathsf{sk}_{\text{iss}} \cdot X_i$, which is a BLS signature on $\mathsf{pk}_{\text{eph},i}$. Finally, the correctness of the Spend and Refund follows directly from the correctness of the underlying digital 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 $\tilde{\sigma}_i = (k \cdot \mathsf{sk}_{\text{iss}}) \cdot X_i$ where $\mathsf{com}_k = k \cdot \mathsf{pk}_{\text{iss}}$. Moreover, upon successful completion of Π-Execute, $\mathcal{C}$ learns the masking key k. Hence, using its knowledge of (r,k), $\mathcal{C}$ can unmask each pre-token by computing $\alpha_i := (r \cdot k)^{-1} \cdot \tilde{\sigma}_i = \mathsf{sk}_{\text{iss}} \cdot X_i$, which is a BLS signature on $\mathsf{pk}_{\text{eph},i}$.

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

Finally, if the spend capacity for pk_{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 pk_{ref}, the public key selected by $\mathcal{C}$ at the time of issuance. □

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 pk_{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 pk_{iss}, we mint a total of $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\text{iss}}] + \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\text{iss}}]$ units of ETH.

Our protocol also burns ETH in two cases as well. First, during issuance, we burn ℓ units of ETH for issuing ℓ tokens. Second, while retiring pk_{iss} we burn Δ units of ETH, where:

$$\Delta := \max(0, \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\text{iss}}] + \mathcal{E}.\mathsf{Nref}[\mathsf{pk}_{\text{iss}}] - \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}])$$

The quantity Δ above precisely captures the excess ETH that the protocol mints on behalf of issuer pk_{iss} beyond what is burnt during on-chain issuance associated with pk_{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 pk_{iss}. Note that in our protocol:

(i) for issuer pk_{iss}, we issue at most half of its escrowed amount, i.e., $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\text{iss}}]/2$;

(ii) we allow spends of at most $\mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}]$ spends, i.e., $\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\text{iss}}] \leq \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\text{iss}}]/2$; and

(iii) we allow refunds only for issued tokens, i.e., $\mathcal{E}.\mathsf{Refund}[\mathsf{pk}_{\text{iss}}] \leq \mathcal{E}.\mathsf{Niss}[\mathsf{pk}_{\text{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\text{iss}}]/2$.

Combining these bounds, we obtain

$$\mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\text{iss}}] + \mathcal{E}.\mathsf{Refund}[\mathsf{pk}_{\text{iss}}] \leq \mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\text{iss}}]$$

Since $\Delta \leq \mathcal{E}.\mathsf{Nspent}[\mathsf{pk}_{\text{iss}}] + \mathcal{E}.\mathsf{Refund}[\mathsf{pk}_{\text{iss}}]$, and the issuer pk_{iss} escrows $\mathcal{E}.\mathsf{esc}[\mathsf{pk}_{\text{iss}}]$ units of ETH, we can always burn Δ units of ETH while retiring pk_{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 $\mathcal{G}_{\mathsf{unlink}}^{\mathsf{BAT},\mathcal{E},\mathcal{A}}$ (see Fig. 5). First, note that we mask each in a

session independent of other tokens. Therefore, it is sufficient to analyze $\mathcal{A}$'s view specific to the challenge token.

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

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

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 *selective* adversary. Specifically, we require an adversary $\mathcal{A}$ to be selective in the following sense: for each invocation of the issuance oracle $\mathcal{O}$-MIssue, $\mathcal{A}$ commits in advance to whether it will invoke the execution oracle $\mathcal{O}$-Execute for that session. This selectivity restriction is an artifact of our security proofs, and we are not aware of any attack that an *adaptive* adversary could mount against our protocol [16, 64]. We further conjecture that our protocol can be proven secure against adaptive adversaries in the Generic Group Model [67].

Lemma 6.5 (Issuer fairness)**.** *Assuming the hardness of one-more computational Diffie-Hellman (*omCDH*) assumption (Def. A.1), the* BAT *protocol in Fig. 7 ensures issuer-fairness as per Def. 4.2 against any selective adversary.*

Proof of Issuer-fairness (Lemma 6.5). For the sake of contradiction, let $\mathcal{A}$ be a selective adversary that breaks the issuer-fairness protocol. We will then use $\mathcal{A}$ to build an omCDH adversary $\mathcal{B}$ as follows.

Simulating setup. $\mathcal{B}$ upon receiving the omCDH challenge $(\llbracket 1 \rrbracket_1, \llbracket z_0 \rrbracket_2, \{\llbracket z_i \rrbracket_1\}_{i \in [0, q_{\mathsf{H}}]})$, sets the issuer's public key $\mathsf{pk}_{\mathsf{iss}} := \llbracket z_0 \rrbracket_2$, which implicitly sets $\mathsf{sk}_{\mathsf{iss}} := z_0$. Let q_{H} be the upper bound on the number of H queries $\mathcal{A}$ makes.

Simulating random oracle $\mathsf{H}(\cdot)$. $\mathcal{B}$ maintains a list L_{H} of all H queries so far. When $\mathcal{A}$ queries $\mathsf{H}(\cdot)$ on input an ephemeral public key X_θ:

1. $\mathcal{B}$ checks if $X_\theta \in L_{\mathsf{H}}$. If so, it returns the stored value.
2. Otherwise, $\mathcal{B}$ sets $L_{\mathsf{H}}[X_\theta] = \llbracket z_\theta \rrbracket_1$, and outputs $L_{\mathsf{H}}[X_\theta]$, and updates $\theta := \theta + 1$

Book keeping. $\mathcal{B}$ uses $N_{\mathsf{iss}} := 0$ and $N_{\mathsf{ex}} := 0$ to maintain the number of tokens for which $\mathcal{A}$ has run the $\mathcal{O}$-MIssue and $\mathcal{O}$-Execute protocol, respectively. For each invocation of $\mathcal{O}$-MIssue with request for ℓ tokens, $\mathcal{B}$ updates $N_{\mathsf{iss}} := N_{\mathsf{iss}} + \ell$. Similarly, for each invocation of $\mathcal{O}$-Execute with ℓ tokens, $\mathcal{B}$ updates $N_{\mathsf{ex}} := N_{\mathsf{ex}} + \ell$.

Let N^*_{iss} and N^*_{ex} be the values of N_{iss} and N_{ex}, at the time when $\mathcal{A}$ breaks the issuer-fairness, respectively. Next, depending on N^*_{iss} and N^*_{ex}, we will consider two mutually exclusive and exhaustive cases.

Case-I: $N^*_{\mathsf{iss}} = N^*_{\mathsf{ex}}$**.** Intuitively, this case captures the scenario in which $\mathcal{A}$ invokes $\mathcal{O}$-Execute for all sessions for which it invoked $\mathcal{O}$-MIssue. In this case, $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

Simulating $\mathcal{O}$-MIssue *protocol.* Consider an invocation of $\mathcal{O}$-MIssue protocol with ℓ tokens. Let sid be its session id. Let $\{X_i\}_{i \in [\ell]} \in \mathbb{G}^\ell$ be the messages $\mathcal{B}$ receives from $\mathcal{A}$. $\mathcal{B}$ does the following:

1. Update $N_{\mathsf{iss}} := N_{\mathsf{iss}} + \ell$
2. For each $i \in [\ell]$, query the $\mathcal{O}$-CDH oracle on input X_i to receive $z_0 \cdot X_i$
3. Honestly sample $k_{\mathsf{sid}} \leftarrow\$ \mathbb{Z}_p$, compute $\mathsf{com}_k := k_{\mathsf{sid}} \cdot \mathsf{pk}_{\mathsf{iss}}$, and store k_{sid} for the session.
4. Send $(\{\tilde{\sigma}_i := k_{\mathsf{sid}} \cdot (z_0 \cdot X_i)\}_{i \in [\ell]}, \mathsf{com}_k)$ to $\mathcal{A}$.

Simulating $\mathcal{O}$-Execute *protocol.* For any invocation of $\mathcal{O}$-Execute using session id sid, $\mathcal{B}$ follows the honest protocol to post k_{sid} on-chain.

Solving omCDH. Observe that to win the issuer-fairness game, $\mathcal{A}$ needs to output $N^*_{\mathsf{iss}} + 1$ BLS signatures. Let $\{(\mathsf{pk}_{\mathsf{eph},j}, \alpha_j)\}_{j \in [N^*_{\mathsf{ex}}+1]}$ be these message signature pairs.

Without loss of generality, we can assume that $\mathcal{A}$ has queried $\mathsf{H}(\mathsf{pk}_{\mathsf{eph},j})$, i.e., $Y_j = \mathsf{H}(\mathsf{pk}_{\mathsf{eph},j}) = \llbracket z_j \rrbracket_1$, where $\llbracket z_j \rrbracket_1$ is an element of the omCDH challenge. Now, since each α_j is a valid BLS signature, we have $\alpha_j := z_0 \cdot \llbracket z_j \rrbracket_1$. This implies that $\mathcal{A}$ outputs $N^*_{\mathsf{ex}} + 1 = N^*_{\mathsf{iss}} + 1$ CDH solutions.

Now, let's calculate the number of $\mathcal{O}$-CDH queries $\mathcal{B}$ makes while interacting with $\mathcal{A}$. For each invocation of $\mathcal{O}$-MIssue with ℓ tokens, $\mathcal{B}$ queries $\mathcal{O}$-CDH on ℓ inputs. Therefore, $\mathcal{B}$ queries $\mathcal{O}$-CDH N^*_{iss} times in total.

Thus, using the $N^*_{\mathsf{iss}} + 1$ CDH solution that $\mathcal{A}$ outputs, and the remaining $q_{\mathsf{H}} - 1 - N^*_{\mathsf{iss}}$ remaining $\mathcal{O}$-CDH queries, $\mathcal{B}$ can solve the omCDH problem with parameter q_{H} using only $q_{\mathsf{H}} - 1$ CDH queries to the $\mathcal{O}$-CDH oracle.

Case-II: $N^*_{\mathsf{iss}} > N^*_{\mathsf{ex}}$**.** This case captures the scenario where $\mathcal{A}$ does not invoke $\mathcal{O}$-Execute for all the sessions it invoked $\mathcal{O}$-MIssue for, i.e., there exists at least one $\mathcal{O}$-MIssue session, for which $\mathcal{A}$ has not invoked the $\mathcal{O}$-Execute oracle. Note that in this case, $\mathcal{B}$ we describe for Case-I can no longer use the $N^*_{\mathsf{ex}} + 1$ BLS signatures output by $\mathcal{A}$ to compute CDH solutions for $N^*_{\mathsf{iss}} + 1$.

For this case, we assume that $\mathcal{A}$ is *selective*, i.e., for each invocation of $\mathcal{O}$-MIssue, $\mathcal{A}$ specifies whether it will invoke $\mathcal{O}$-Execute for the session or not. More precisely, for the i-th invocation of $\mathcal{O}$-MIssue, $\mathcal{A}$ specifies a bit ν_i, where $\nu_i = 1$ implies that $\mathcal{A}$ will invoke $\mathcal{O}$-Execute for the i-th session, and $\nu_i = 0$, otherwise.

Simulating $\mathcal{O}$-MIssue protocol. Consider the i-th issuance session for ℓ tokens and session id sid. Let $\{X_i\}_{i\in[\ell]} \in \mathbb{G}^\ell$ be the messages $\mathcal{B}$ receives from $\mathcal{A}$.

When $\nu_i = 1$, $\mathcal{B}$ simulates the $\mathcal{O}$-MIssue as in case-I. Alternatively, when $\nu_i = 0$, $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

1. Update $N_{\mathsf{iss}} := N_{\mathsf{iss}} + \ell$.
2. Sample $\mu_{\mathsf{sid}} \leftarrow\!\!\$\ \mathbb{Z}_p$, implicitly defining $\mu_{\mathsf{sid}} = z_0 \cdot k_{\mathsf{sid}}$ for some $k_{\mathsf{sid}} \in \mathbb{Z}_p$.
3. Compute $\mathsf{com}_k := [\![\mu_{\mathsf{sid}}]\!]_2$.
4. Send $(\{\tilde{\sigma}_i := \mu_{\mathsf{sid}} \cdot X_i\}_{i\in[\ell]}, \mathsf{com}_k)$ to $\mathcal{A}$.

Note that since $\mathcal{B}$ samples μ_{sid} uniformly at random, $\mathcal{A}$'s view is identically distributed for both $\nu = 1$ and $\nu = 0$.

Now, it is easy to see that $\mathcal{B}$ uses one CDH query for each of the tokens that $\mathcal{A}$ receives upon invoking $\mathcal{O}$-Execute. Therefore, to see that when $\mathcal{A}$ outputs $N^*_{\mathsf{ex}} + 1$ tokens to win the game, $\mathcal{B}$ can use them, and the remaining $q_{\mathsf{H}} - 1 - N^*_{\mathsf{ex}}$ CDH queries to solve the omCDH problem with parameter q_{H} using at most $q_{\mathsf{H}} - 1$ CDH oracle queries. □

7 Implementation and Evaluation

We implemented our BAT schemes (Fig. 7 and Fig. 9) in Rust, and evaluated the performance of each phase. We report execution time and communication cost, and break them down into client-side, issuer-side, and on-chain (environment-side) costs. While BAT on-chain components can be implemented as smart contracts, we implement our on-chain component natively.

We henceforth refer to our protocol in Fig. 7 as Protocol A, and the protocol that facilitates non-interactive refunds in Fig. 9 as Protocol B. Protocol A and Protocol B use different cryptographic instantiations.

For Protocol A, we use the BLS12-381 curve [9, 24] for issuer signatures and Schnorr signatures [65] over the edwards25519 curve [14] for clients' signatures on the ephemeral keys. We instantiate Protocol B over the BN254 curve [10] for both the issuer's and the client's 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[1]) 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 [47] 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 [66]. We use Spartan instead of Groth16 [41] 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 [42] 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 and reasonable 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.

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 [28]. 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

[1]The Grumpkin curve [59] is a Weierstrass elliptic curve designed for SNARK-efficient group operations, specifically 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.

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.

8 Discussion for Practical Deployments

Token denomination. We describe 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.

Non-interactive refunds and unlinkability. In the alternate BAT scheme we describe in Fig. 9, 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.

Refund haircut. In practice, a small haircut should be applied to the refunds to prevent abuse of the system by an adversary that issues a large number of tokens only to refund them later.

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.

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] Amit Agarwal, Kushal Babel, Sourav Das, and Babak Poorebrahim Gilkalaye. Time-lock encrypted storage for blockchains. Cryptology ePrint Archive, Paper 2025/2048, 2025.

[2] Amit Agarwal, Kushal Babel, Sourav Das, Babak Poorebrahim Gilkalaye, Arup Mondal, Benny Pinkas, Peter

Rindal, and Aayush Yadav. Weighted batched threshold encryption with applications to mempool privacy. Cryptology ePrint Archive, Paper 2025/2115, 2025.

[3] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, pages 69–100, Cham, 2025. Springer Nature Switzerland.

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

[5] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2499–2516. IEEE, 2023.

[6] Foteini Baldimtsi, Kostas Kryptos Chalkias, Varun Madathil, and Arnab Roy. SoK: Privacy-preserving transactions in blockchains. Cryptology ePrint Archive, Paper 2024/1959, 2024.

[7] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. Anonymous transferable E-cash. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 101–124, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer Berlin Heidelberg, Germany.

[8] Foteini Baldimtsi, Lucjan Hanzlik, Quan Nguyen, and Aayush Yadav. Non-interactive anonymous tokens with private metadata bit. Cryptology ePrint Archive, Report 2025/430, 2025.

[9] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, September 2004.

[10] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331, Kingston, Ontario, Canada, August 11–12, 2006. Springer Berlin Heidelberg, Germany.

[11] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[12] Fabrice Benhamouda, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Publicly verifiable anonymous tokens with private metadata bit. Cryptology ePrint Archive, Report 2022/004, 2022.

[13] Ferenc Beres, Istvan A. Seres, Andras A. Benczur, and Mikerah Quintyne-Collins. Blockchain is Watching You: Profiling and Deanonymizing Ethereum Users . In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 69–78, Los Alamitos, CA, USA, August 2021. IEEE Computer Society.

[14] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142, Nara, Japan, September 28 – October 1, 2011. Springer Berlin Heidelberg, Germany.

[15] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2002.

[16] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 2004.

[17] Dan Boneh, Benedikt Bünz, Kartik Nayak, Lior Rotem, and Victor Shoup. Context-dependent threshold decryption and its applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 506–538. Springer, 2025.

[18] Dan Boneh, Evan Laufer, and Ertem Nusret Tas. Batch decryption without epochs and its application to encrypted mempools. *Cryptology ePrint Archive*, 2025.

[19] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer Berlin Heidelberg, Germany.

[20] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.6*, 2023.

[21] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[22] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. Beast-mev: Batched threshold encryption with silent setup for mev prevention. *Cryptology ePrint Archive*, 2025.

[23] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyan Qu. {BEAT-MEV}: Epochless approach to batched threshold encryption for {MEV} prevention. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3457–3476, 2025.

[24] Sean Bowe. Bls12-381: New zk-snark elliptic curve construction. https://electriccoin.co/blog/new-snark-curve/, Mar 11 2017. Electric Coin Company Blog.

[25] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 423–443, Kota Kinabalu, Malaysia, February 10–14, 2020. Springer, Cham, Switzerland.

[26] Vitalik Buterin, Jacob Illum, Matthias Nadler, Fabian Schär, and Ameen Soleimani. Blockchain privacy and regulatory compliance: Towards a practical equilibrium. *Blockchain: Research and Applications*, 5(1):100176, 2024.

[27] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[28] Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. Curve trees: Practical and transparent {Zero-Knowledge} accumulators. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4391–4408, 2023.

[29] Melissa Chase, F. Betül Durak, and Serge Vaudenay. Anonymous tokens with stronger metadata bit hiding from algebraic MACs. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 418–449, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[30] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 199–203, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA.

[31] David Chaum. Showing credentials without identification transferring signatures between unconditionally unlinkable pseudonyms. In Jennifer Seberry and Josef Pieprzyk, editors, *Advances in Cryptology – AUSCRYPT'90*, volume 453 of *Lecture Notes in Computer Science*, pages 246–264, Sydney, Australia, January 8–11, 1990. Springer Berlin Heidelberg, Germany.

[32] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327, Santa Barbara, CA, USA, August 21–25, 1990. Springer, New York, USA.

[33] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[34] Arka Rai Choudhuri, Sanjam Garg, Guru Vamsi Policharla, and Mingyuan Wang. Practical mempool privacy via one-time setup batched threshold encryption. In *34th USENIX Security Symposium*, 2025.

[35] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*, pages 910–927. IEEE, 2020.

[36] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018(3):164–180, July 2018.

[37] Benjamin E. Diamond. Many-out-of-many proofs and applications to anonymous zether. In *2021 IEEE Symposium on Security and Privacy*, pages 1800–1817, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[38] Ádám Ficsór, István András Seres, Yuval Kogman, and Lucas Ontivero. Wabisabi: Centrally Coordinated CoinJoins with Variable Amounts. *Cryptoeconomic Systems*, 1(2), oct 22 2021. https://cryptoeconomicsystems.pubpub.org/pub/ficsor-wabisabi-coordinated.

[39] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. tlock: Practical timelock encryption from threshold BLS. Cryptology ePrint Archive, Paper 2023/189, 2023.

[40] Pranav Garimidi, Joseph Bonneau, and Lioba Heimbach. On the limits of encrypted mempools, July 2025. a16z Crypto.

[41] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

[42] Mike Hamburg. Decaf: Eliminating cofactors through point compression. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 705–723, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany.

[43] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub. In *ISOC Network and Distributed System Security Symposium – NDSS 2017*, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society.

[44] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain Bitcoin transactions. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 43–60, Christ Church, Barbados, February 26, 2016. Springer Berlin Heidelberg, Germany.

[45] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, page 61–70, New York, NY, USA, 2005. Association for Computing Machinery.

[46] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in zcash. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 463–477, 2018.

[47] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[48] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous tokens with private metadata bit. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 308–336, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[49] Thomas Lloyd, Daire O'Broin, and Martin Harrigan. Emergent outcomes of the vetoken model. In *2023 IEEE international conference on omni-layer intelligent systems (COINS)*. IEEE, 2023.

[50] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. {VoteAgain}: A scalable coercion-resistant voting system. In *29th USENIX security symposium (USENIX Security 20)*, pages 1553–1570, 2020.

[51] Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world. Bitcointalk Forum, 2013.

[52] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(2):105–121, April 2018.

[53] Sarah Meiklejohn and Claudio Orlandi. Privacy-enhancing overlays in Bitcoin. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *FC 2015 Workshops*, volume 8976 of *Lecture Notes in Computer Science*, pages 127–141, San Juan, Puerto Rico, January 30, 2015. Springer Berlin Heidelberg, Germany.

[54] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140, 2013.

[55] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[56] Michele Orrù, Stefano Tessaro, Greg Zaverucha, and Chenzhi Zhu. Oblivious issuance of proofs. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part IX*, volume 14928 of *Lecture Notes in Computer Science*, pages 254–287, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[57] Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado cash privacy solution version 1.4, 2019. Accessed: 2026-02-01.

[58] Privacy and Scaling Explorations. Introduction to MACI, Referenced Jan. 2026.

[59] Aztec 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.

[60] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022.

[61] Michael K Reiter and Kenneth P Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):986–1009, 1994.

[62] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565, Gold Coast, Australia, December 9–13, 2001. Springer Berlin Heidelberg, Germany.

[63] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical decentralized coin mixing for Bitcoin. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 345–364, Wroclaw, Poland, September 7–11, 2014. Springer, Cham, Switzerland.

[64] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer Berlin Heidelberg, Germany.

[65] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[66] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[67] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 256–266. Springer, 1997.

[68] Shutter Network. Shutter network introduces plan for first encrypted mempool on ethereum.

[69] Tjerand Silde and Martin Strand. Anonymous tokens with public metadata and applications to private contact tracing. In Ittay Eyal and Juan A. Garay, editors, *FC 2022: 26th International Conference on Financial Cryptography and Data Security*, volume 13411 of *Lecture Notes in Computer Science*, pages 179–199, Grenada, May 2–6, 2022. Springer, Cham, Switzerland.

[70] Sora Suegami, Shinsaku Ashizawa, and Kyohei Shibano. Constant-cost batched partial decryption in threshold encryption. *Cryptology ePrint Archive*, 2024.

[71] Ertem Nusret Tas, István András Seres, Yinuo Zhang, Márk Melczer, Mahimna Kelkar, Joseph Bonneau, and Valeria Nikolaenko. Atomic and fair data exchange via blockchain. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 3227–3241, 2024.

[72] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. UTT: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Report 2022/452, 2022.

[73] Nicolas van Saberhagen. CryptoNote v 2.0. https://cryptonote.org/whitepaper.pdf, 2013. The foundational protocol for Monero.

[74] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. A flash (bot) in the pan: measuring maximal extractable value in private pools. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 458–471, 2022.

[75] John Wilander. Introducing private click measurement, pcm. https://webkit.org/blog/11529/introducing-private-click-measurement-pcm/, 2021. Accessed: 2024-04-29.

A Additional Preliminaries

Computational assumptions. We make the following computational assumptions for our scheme.

Definition A.1 (One-more CDH)**.** The one-more computational Diffie-Hellman (omCDH) assumption holds relative to

Game $\mathcal{G}^{\mathcal{A}}_{\mathsf{omCDH}}(1^\lambda, k)$:

$\tilde{\ell} := 0$

$\{z_0, z_1, \ldots, z_k\} \leftarrow\!\!\$\ \mathbb{Z}_p$

$\{\tilde{Z}_i\}_{i\in[k]} \leftarrow \mathcal{A}^{O\text{-}\mathsf{CDH}}([\![z_0]\!]_1, [\![z_0]\!]_2, \{[\![z_i]\!]_1\}_{i\in[k]})$

require $\tilde{\ell} < k \wedge \forall i \in [k]\ \ \tilde{Z}_i = [\![z_0 \cdot z_i]\!]_1$

return 1

Oracle O-$\mathsf{CDH}(H)$:

require $H \in \mathbb{G}_1$

$\tilde{\ell} := \tilde{\ell} + 1$

return $z_0 \cdot H$

Figure 8: The one-more computational Diffie-Hellman game $\mathcal{G}^{\mathcal{A}}_{\mathsf{omCDH}}$.

GGen, if for all PPT adversary $\mathcal{A}$, for all $k \in \mathbb{N}$, the following probability is negligible:

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{omCDH}}(1^\lambda, k) \Rightarrow 1]$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, [\![1]\!]_1, [\![1]\!]_2, e) \leftarrow \mathsf{GGen}(1^\lambda)$.

A.1 Digital signatures

Definition A.2 (Digital signature)**.** A digital signature scheme consists of the following four PPT algorithms $\mathsf{DS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$, defined as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. The setup algorithm that takes as input a security parameter λ and outputs system parameters pp. These parameters include the description of the message space and the key space. All algorithm below implicitly takes pp as input.
- $\mathsf{KGen}() \to (\mathsf{sk}, \mathsf{pk})$: The key generation algorithm outputs a pair of keys $(\mathsf{sk}, \mathsf{pk})$, where sk is the secret (signing) key and pk is the public (verification) key.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: The signing algorithm that takes the secret key sk and a message m, and outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to 0/1$: The verification algorithm is a deterministic algorithm that takes the public key pk, a message m, and a signature σ, and outputs a bit $b \in \{0,1\}$. The output $b = 1$ indicates that the signature is valid, and $b = 0$ indicates that it is invalid.

A digital signature scheme is secure if it satisfies *correctness* and *unforgeability* properties. We refer the reader to [20] for formal definitions of these properties.

In this paper, we will be concretely using the bilinear pairing based BLS signature from [19] and its blinded version from [15]. We briefly describe them next.

BLS blind signature [15, 31]. Recall from [31], in a blind signature scheme, a client seeks to get a signature on a message m where the signer learns no information about m.

To obtain a blind signature on a message $m \in \mathcal{M}$, the client samples a random blinding factor $r \leftarrow \mathbb{Z}_p$ and computes the blinded message $\tilde{m} = r \cdot \mathsf{H}(m) \in \mathbb{G}_2$ and sends $\tilde{m}$ to the signer. The signer computes a blind signature $\tilde{\sigma} = \mathsf{sk} \cdot \tilde{m} = (r \cdot \mathsf{sk}) \cdot \mathsf{H}(m) \in \mathbb{G}_2$ and returns it to the user. The user then unblinds the signature by computing $\sigma = (1/r) \cdot \tilde{\sigma} = \mathsf{sk} \cdot \mathsf{H}(m) \in \mathbb{G}_2$, which is a valid BLS signature on m. Similar to the BLS signature, the security of BLS blind signature also relies on the hardness of the co-CDH assumption in the ROM [15].

A.2 Commitment Schemes

Definition A.3 (Commitment Scheme)**.** A commitment scheme is a tuple of four PPT algorithms $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ defined as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}_{\mathsf{com}}$. The setup algorithm outputs public parameters $\mathsf{pp}_{\mathsf{com}}$, which are given as an implicit input to all subsequent algorithms.
- $\mathsf{Commit}(m) \to (c, d)$. The commitment algorithm takes a message m and outputs a commitment c together with opening information d.
- $\mathsf{Open}(c, d) \to m$. The opening algorithm outputs (m, d).
- $\mathsf{Verify}(c, m, d) \to 0/1$. The verification algorithm outputs 1 if (m, d) is a valid opening of c, and 0 otherwise.

A commitment scheme must satisfy correctness, hiding, and binding properties. We refer the reader to standard references [20] for formal definitions.

A.3 Non-Interactive Zero-Knowledge Proofs

We use non-interactive zero-knowledge (NIZK) proof systems for proving statements about committed values.

Definition A.4 (NIZK Proof System)**.** A non-interactive zero-knowledge proof system for a relation $\mathcal{R}$ consists of four PPT algorithms $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ defined as follows:

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to \mathsf{crs}$. The setup algorithm outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$. The proving algorithm takes a statement, witness tuple $(x, w) \in \mathcal{R}$, and outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 0/1$. The verification algorithm checks if π is a valid proof for statement x.
- $\mathsf{Sim}(\mathsf{crs}, x) \to \pi$. The simulator generates a proof for statement x without a witness.

A NIZK proof system must satisfy completeness, soundness, and zero-knowledge. We refer the reader to standard references [20] for formal definitions of these properties.

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

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

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

$\mathsf{IssKGen}, \mathsf{Register}, \mathsf{Retire}$

Same as Figure 7

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

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

$\mathsf{TokGen}, \mathsf{Spend}$

Same as Figure 7

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

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

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

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

Figure 9: BAT protocol with compact issuance and refunds based on revealing the seed. $\mathcal{C}$: denotes actions by the client, I : by the issuer, and $\mathcal{E}$: by the environment $\mathcal{E}$. Additional parts compared to Figure 7 are marked in blue.