Efficiently-Thresholdizable Selective Batched Identity Based Encryption, with Applications

Amit Agarwal
University of Illinois Urbana-Champaign (UIUC)

Rex Fernando
Aptos Labs

Benny Pinkas
Aptos Labs, Bar Ilan University

October 6, 2024

Abstract

We propose a new cryptographic primitive called "selective batched identity-based encryption" (Selective Batched IBE) and its thresholdized version. The new primitive allows encrypting messages with specific identities and batch labels, where the latter can represent, for example, a block number on a blockchain. Given an arbitrary subset of identities for a particular batch, our primitive enables efficient issuance of a single decryption key that can be used to decrypt all ciphertexts having identities that are included in the subset while preserving the privacy of all ciphertexts having identities that are excluded from the subset. At the heart of our construction is a new technique that enables public aggregation (i.e. without knowledge of any secrets) of any subset of identities, into a succinct digest. This digest is used to derive, via a master secret key, a single *succinct* decryption key for all the identities that were digested in this batch. In a threshold system, where the master key is distributed as secret shares among multiple authorities, our method significantly reduces the communication (and in some cases, computation) overhead for the authorities. It achieves this by making their costs for key issuance independent of the batch size.

We present a concrete instantiation of a Selective Batched IBE scheme based on the KZG polynomial commitment scheme by Kate et al. (Asiacrypt'10) and a modified form of the BLS signature scheme by Boneh et al. (Asiacrypt'01). The construction is proven secure in the generic group model (GGM).

In a blockchain setting, the new construction can be used for achieving mempool privacy by encrypting transactions to a block, opening only the transactions included in a given block and hiding the transactions that are not included in it. With the thresholdized version, multiple authorities (validators) can collaboratively manage the decryption process. Other possible applications include scalable support via blockchain for fairness of dishonest majority MPC, and conditional batched threshold decryption that can be used for implementing secure Dutch auctions and privacy preserving options trading.

Contents

1 Introduction

This paper studies problems related to efficient batch decryption of identity-based encryption (IBE), and particularly threshold batch decryption in a blockchain setting. The solutions we pro-

pose support *conditional* batched threshold decryption of arbitrary subsets of ciphertexts, and can be used to provide a scalable support via blockchain for fair dishonest majority MPC.

Both standard threshold encryption and threshold identity-based encryption have received much attention in the blockchain setting. For example, (standard) threshold encryption has been used to achieve mempool privacy [BO22] (further discussed below), and threshold IBE has been used to achieve "encryption to the future" [Cam+21; GMR23; Döt+23; Cer+23]. With encryption to the future, the idea is that if the validators of a particular chain have shared an IBE master secret key, then during a block they can release decryption keys for the ID which is that block's number or the time on which that block is published. Anyone who wants to encrypt a message to be decrypted at a specific block in the future, or at a specific time, can use the IBE public key with the appropriate ID. However, a major drawback of this technique is that it decrypts *all* messages encrypted to a specific block, and does not enable to dynamically choose which of these messages to decrypt.

A possible solution to the aforementioned all-or-nothing decryption problem is to encrypt each message using a separate key. In that case, if there are B ciphertexts to be decrypted, and n holders of the secret key shares (i.e., the validators, in the blockchain setting), with a decryption threshold of $\Omega(n)$, standard threshold decryption requires these parties to do $O(nB)$ computation and communication (per party that needs to decrypt the message). This is especially problematic when trying to achieve mempool privacy, on which we elaborate next.

The goal of mempool privacy stems from the way that transactions are submitted to blockchains. Before transactions are finalized, they are held in a *memory pool (mempool)*, which is publicly readable. That a mempool is public is inherent: it must be readable by all the validators so that they are able to build the next block, and the design of blockchains as permissionless networks allows for anyone to run a validator. The fact that the mempool is public and contains information on what transactions will be in future blocks makes it ripe for exploitation. Such exploitation is widespread, and has been termed *miner-extractable value*, or MEV.[1] The main technique for combating this type of abuse is to encrypt the transactions in the mempool, and to only decrypt the transactions in a block after the block has been finalized (see, e.g., [Kav+23] and references within). That way, although the mempool is still public, it is opaque. In addition, since a block has limited capacity and might not include all transactions submitted to it, it is crucial that the decryption process can choose to decrypt any subset of the transactions encrypted to the block, while keeping hidden the transactions outside of this subset. On the other hand, independent threshold decryption of each transaction incurs a total overhead of $O(nB)$ communication and computation, which is often too high for modern blockchains that are built to achieve high throughput and very low latency. Several recent works have studied this problem and have proposed solutions (see [Cho+24] and references within). Specifically, [Cho+24] attempts to do better than $O(nB)$ communication per validator; they end up with an *online* phase which has $O(n)$ communication per validator, but rely on an *expensive offline per-block interactive setup phase*.[2]

[1] This concept was introduced in [Dai+20] which explored transaction reordering and front-running in decentralized exchanges (DEXs) on the Ethereum blockchain. The term MEV refers to the additional profits validators can extract by reordering, censoring, or including transactions in a specific way within a block. This result was groundbreaking because it highlighted a significant and underexplored vulnerability in decentralized finance (DeFi) systems. This result has since been highly influential in research on blockchain economics, consensus protocols, and DeFi security.

[2] As was previously mentioned, IBE for "encryption to the future" can be used as a way around the $O(nB)$ work per validator. That is, in each round, the validators collaboratively compute and publish a *single decryption key* corresponding to the current block number. This single key can be used to decrypt an arbitrary number of ciphertexts that have

1.1 Our results

We introduce and construct a new primitive, which we call **Selective Batched Identity Based Encryption** (Selective Batched IBE). This new notion solves the efficiency problem described above for threshold decryption, and also has several other interesting applications, in both the threshold and the non-threshold setting. Our new notion works as follows.

- As with standard IBE, encryptions are done with respect to some ID. In addition, an encryption also specifies a *batch label*.
- Any set of IDs, up to a pre-specified maximum batch size, can be *publicly aggregated* to produce a succinct digest.
- A succinct decryption key can be computed from this digest, a specified batch label, and a master secret key. This computation is done in *constant time* relative to the batch size. The key can then be used to decrypt any ciphertext that was encrypted with respect to any ID in the digest and the matching batch label.
- Optionally, the decryption key computation can be thresholdized.

We have the following contributions in this work.

- In Section 4, we formally define the notion of Selective Batched Identity Based Encryption (Selective Batched IBE).
- In Section 5, we provide an efficient construction for Selective Batched IBE based on Type-3 pairings and prove its security in the Generic Group Model (GGM).
- In Section 6, we discuss different optimizations which can be used to speed up the digestion and decryption process in our basic construction and an efficient way to add non-malleability for mempool privacy application.
- In Appendix A, we define a threshold verison of Selective Batched IBE, called Thresholdizable Selective Batched IBE , and efficiently extend our non-threshold construction to the threshold version.
- In Section 7, we discuss the implementation of our scheme and analyze its concrete performance, both in the threshold and non-threshold setting.
- In Section 1.2, we provide many interesting applications of our primitive.

1.2 Cryptographic Applications

Our construction opens up an avenue for a wide range of applications. We begin by describing an application to fair dishonest-majority MPC via blockchain, and then describe other applications of conditional batched threshold decryption.

been encrypted towards this block. Notice, however, that this decryption is *all-or-nothing* and opens all ciphertexts with the corresponding ID. This is not a suitable solution since a block might not be able to include all transactions that were submitted to it. Therefore it must be possible to decrypt any subset of the transactions that submitted to a block, while keeping the other transactions hidden. As such, threshold IBE fails to solve the efficiency problem in the mempool privacy case.

Fair Dishonest majority MPC via Blockchain. Secure Multi-party Computation (MPC) allows two or more parties to compute any public function over their privately-held inputs, without revealing any information beyond the result of the computation. An extremely desirable property of such a MPC protocol is *fairness*, namely ensuring that either all parties learn the output or no one does. Unfortunately, in a dishonest majority setting (where the adversary can actively corrupt more than half the number of parties), achieving fairness is impossible [Cle86] in general in the standard MPC model [Gol09]. Yet this property is crucial for applications like sealed-bid auctions and contract signing, where information asymmetry can be exploited by a malicious party.

An intriguing method introduced in [Cho+17] suggests achieving fairness through witness encryption (WE) and public bulletin boards such as blockchains. The parties execute an unfair MPC protocol, completely off-chain, to compute a WE ciphertext of the output, rather than the output itself. The construction ingeniously employs a "release token" as a witness for decrypting the WE ciphertext. First, each party generates a secret share of the release token, ensuring that the token value is shared between all parties. The statement used for encrypting the output with WE is designed so that its valid witness is a *proof* of publishing the full release token on a public bulletin board. This proof can correspond, for example, for a signature of the blockchain on all shares of the release token. To decrypt the output, a party must have all shares of the release token be *published* on the blockchain. As a result, all shares are available to all parties, not only to the last party to provide its share.

While the result in [Cho+17] is theoretical, requiring general-purpose WE which is prohibitively inefficient, we observe that a slightly modified version of their idea can be efficiently instantiated using our Selective Batched IBE primitive. Instead of computing a WE encryption, the MPC computes an encryption to a specific ID that identifies this MPC session, and a specific block label. Additionally, before executing the MPC, the MPC participants deploy a smart contract that stores the public keys of all MPC participants. The smart contract requires that only given a signature from each of these participants on the ID, the output of the MPC computation can be released. (Essentially, the signature of participant i corresponds to the agreement of this participant to the publication of the output.) The blockchain follows this smart contract, and includes this instance ID in the digest of IDs whose decryption is enabled, only if all participants provided their signatures.

This construction has several nice properties. First, it enables a blockchain to support fairness for an arbitrary number of MPC instances, while running between its validators a *single* threshold computation whose overhead is independent of the number of MPC instances. This property is crucial for scaling, since blockchain validators typically already have a high load related to executing transactions and achieving consensus. Second, the validators do not need to decrypt the results of each MPC session. They merely compute (in constant time) the decryption key that enables the decryption of every MPC session for which all release approvals were given. Third, we observe that our specific construction of Selective Batched IBE based on pairing-friendly groups, is compatible with the popular SPDZ MPC framework [Dam+12] which is one of the most efficient MPC protocol for performing general-purpose (unfair) secure computation in a dishonest majority setting. In [SA19; Dal+20], it is shown that the SPDZ framework (which natively works over a field) can be efficiently extended to perform secure computation over groups. In the context of fair MPC application, the parties would be required to securely emulate the encryption procedure of our Selective Batched IBE construction, which requires just 6 group exponentiations, using MPC. Based on the results provided in [Dal+20], we estimate that the overhead of adding fairness to

an unfair two-party SPDZ MPC would be less than 20 ms (resp. 500 ms) when using SPDZ MPC protocol in a LAN (resp. WAN) setting .

Conditional Batched Threshold Decryption. Our construction enables a form of conditional batch threshold decryption of ciphertexts.

- The fact the decryption is *conditional* enables to decide at the last minute which ciphertexts will be decrypted.
- The *threshold* property enables to distribute trust between multiple servers or validators.
- The *batch* property enables scalability, since the threshold computation of the servers is independent of the number of ciphertexts.

Let us elaborate more on the "batch" and the "conditional decryption" properties.

The "batch label" notion In its simplest form, the batch label can correspond to an event that progresses monotonically along a single dimension. The most obvious examples are batch labels corresponding to a block id or to the time. In particular, the latter option implements *time-lock encryption* (see [RSW96] and followup work). This implementation of time-lock encryption is more efficient than the notion based on moderately-hard computation, while trust becomes dependent on the assumption that the server (or a large enough number of the servers in a threshold setting) are not malicious.

A more complex batch label can correspond to a combination of multiple events in different dimensions. For example, it could correspond to the event "(the USD/EUR exchange rate is above 1 OR the GBP/JPY rate is below 200) AND the date is January 1, 2025". The servers responsible for producing the decryption key for batches will only compute the relevant decryption key when this condition holds.

Conditional decryption The server, or group of servers sharing the master secret key, can decide to enable decryption for any arbitrary subset of the ciphertexts that have the same batch label. There are many examples where this subset cannot be predicted in advance, and the decision about which ciphertexts are decrypted is non-monotone. Sample examples are the Dutch auction application described below, and the fair-MPC application. As another example, consider the case where each transaction submitted to a block has a maximal fee that its sender is willing to pay, and an upper bound on the amount of gas that the transaction might consume. A block has limited capacity in terms of gas, and therefore the validators need to solve a knapsack problem in order to find which subset of submitted transactions will maximize their revenues. The decision on which transactions to decrypt is based on the solution to this problem.

Sample applications Let us further explore some applications that our construction can support.

- Secure Dutch auctions on the blockchain: A *Dutch auction* is an auction where the price starts high and gradually decreases, and the first bidder to accept the current price wins. This type of auction helps determine the market value by finding the highest acceptable price, with a process that is transparent and visible to all participants. We would like to implement

this type of auction in a non-interactive manner, while hiding all bids except for the highest one(s).[3]

Suppose that the price of the good for sale, i.e. the item being auctioned, has a price resolution of m values (say, $m = 1000$), and that bidder i wants to bid a price of $b_i \in [1, m]$. This bidder submits m encryptions with batch labels $1, \ldots, m$, where the encryption with label b_i is of a 1 value, and all encryptions with labels greater than b_i are of 0. (The encryptions of labels smaller than b_i can be arbitrary.) All bidders write a smart contract which begins with decrypting all encryptions with label m. If all these encryptions are of 0, then in the next block the encryptions with label $m-1$ are decrypted, etc. This process stops when one of the decrypted values is 1. When this happens, the bidder (or bidders) who encrypted a 1 value for the current label are declared the winners and have to pay a price equal to the current label. No further values are decrypted. It is easy to verify that this process implements the Dutch auction and hides all bids except for the winning ones. In terms of latency, each price point is decrypted in a separate block, but given the existence of low-latency blockchains, such as as Aptos, Sui or Solana, with sub-second block latency, the overall run time of the auction can be sufficiently fast for many applications. As for efficiency, a single blockchain can support a very large number of auctions, since a single threshold computation by the validators enables to decrypt the bids of all relevant auctions, and the decryption itself can be done publicly and does not require threshold computation.

- Mempool privacy: Our result enables users to submit transactions to a specific block, where the details of each transaction are encrypted using its unique ID and a batch label equal to the block number. Once the validators agree on the transactions that will be included in the block, they aggregate the IDs of these transactions to produce a succinct digest. Given this digest, the validators compute a succinct decryption key for this block. The computation of this decryption key is the only procedure requiring access to a secret (namely, the master secret key), and is thus the only operation that must be computed by a threshold computation. Finally, given the succinct decryption key of this block, it is possible to decrypt all transactions that were included in the digest. All other transactions that were submitted to the block but were not included in it, remain hidden.

  We observe that in this specific application, a ciphertext encrypts a *signed* transaction coming from a specific sender address, associated with the public key of the signature on the transaction. While the authors of [Cho+24] use CCA security for achieving non-malleability, we point out that CCA security is actually an *overkill* for the form of non-malleability required here. This is because in the mempool setting, ciphertexts are associated with a signature public key, and the decryption oracle only decrypts messages that are signed. This enables to achieve non-malleability more cheaply, replacing the NIZK proofs of [Cho+24] with a signature scheme. We refer the readers to Section 6.4 for details.

- Options trading: European options are a type of financial derivative that grants the holder the right, but not the obligation, to buy or sell an underlying asset at a predetermined price on a specified date, known as the expiration date. The defining feature of European options is that they can only be exercised on the expiration date, not before. Our construction can

[3] A Dutch auction is roughly the interactive equivalent of the (non-interactive) first-price sealed-bid auction. So a non-interactive implementation of a Dutch auction is also an implementation of a sealed-bid first-price auction.

be used to hide the terms of such options until the expiration date, and only reveal and execute an option if its holder chooses to execute it. More specifically, every option is encrypted with a batch label that is equal to its expiration date, and with an ID that identifies its holder. Before or at the expiration date, the holder of the option must send a signed execution command. A digest of all options which were authorized to be executed is computed. Afterwards, the server that has the master key publishes the corresponding decryption key that enables to decrypt and execute these options.

2 Technical Overview

Standard IBE versus Selective Batched IBE. In a standard Identity-Based Encryption (IBE) scheme, we have a universe $\mathcal{I}$ of public identities (for example, these could be email ids) and users can create a ciphertext ct w.r.t any $\mathsf{id} \in \mathcal{I}$ so that ct can be decrypted given the identity specific-secret key $\mathsf{sk}_{\mathsf{id}}$. These identity-specific secret keys $\mathsf{sk}_{\mathsf{id}}$ are typically issued by an authority (resp. a set of authorities) holding a master secret key msk (resp. shares of msk) which is used to derive the identity-specific secret key $\mathsf{sk}_{\mathsf{id}}$.

We would like to extend this standard IBE to a batched setting where we have a pool of ciphertexts $\{\mathsf{ct}_1, \ldots, \mathsf{ct}_n\}$ and each ct_j is a ciphertext w.r.t. some identity $\mathsf{id}_j \in \mathcal{I}$. Given a *dynamically selected*[4] public batch of identities $S \subseteq \mathcal{I}$, we would like the authority to release a secret key sk_S which is *succinct* (i.e., independent of the size of set S), and which enables the decryption of all ciphertexts ct_j where $\mathsf{id}_j \in S$ while ensuring that all ciphertexts ct_j corresponding to $\mathsf{id}_j \notin S$ remain hidden.

A naive solution to achieve this would be to simply have sk_S be the set of standard IBE secret keys for the individual identities in the batch S, i.e, $\mathsf{sk}_S = \{\mathsf{sk}_j | \mathsf{id}_j \in S\}$. While this works, the secret key sk_S here has size proportional to $|S|$ which is not succinct.

Boneh-Franklin IBE. Our starting point is the (standard) IBE scheme of Boneh-Franklin [BF01], hereafter referred to as BF-IBE. Here, the master secret key msk is the signing key of a BLS signature scheme [BLS01] and $\mathsf{sk}_{\mathsf{id}}$ is simply a BLS signature on id using the signing key msk. Recall that a BLS signature scheme is defined on a pairing-friendly group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p with group operation $+ : \mathbb{G}_i \times \mathbb{G}_i \to \mathbb{G}_i$ and a pairing operation $\circ : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$. For a group $\mathbb{G}_i$ with generator g_i, we will use the notation $[x]_i$ to represent the group element $x \cdot g_i$ in the group $\mathbb{G}_i$ where $x \in \mathbb{Z}_p$. The signer holds a signing key $\mathsf{msk} \in \mathbb{Z}_p$ and publishes a verification key $\mathsf{vk} = [\mathsf{msk}]_2$. The signature on a message $m \in \{0,1\}^*$ is simply $\sigma_m = \mathsf{msk} \cdot H(m) \in \mathbb{G}_1$ where $H : \{0,1\}^* \to \mathbb{G}_1$ is a hash function modeled as Random Oracle. To verify a claimed signature σ on a message m, the following pairing check is performed.

$$[1]_2 \circ \sigma = \mathsf{vk} \circ H(m)$$

Coming back to the BF-IBE construction, the authority holds a BLS signing key msk and uses it to derive an identity-specific secret key $\mathsf{sk}_{\mathsf{id}} := \sigma_{\mathsf{id}} = \mathsf{msk} \cdot H(\mathsf{id})$. To encrypt a message $m \in \mathbb{G}_T$ w.r.t. identity id, the encryptor samples a random $r \leftarrow \mathbb{Z}_p$ and produces the following ciphertext

$$\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (r \cdot [1]_2, r \cdot (\mathsf{vk} \circ H(\mathsf{id})) + m)$$

[4] By "dynamic", we mean that the subset S of identities can be selected *after* the ciphertexts have been created.

Given an identity secret key $\mathsf{sk}_{\mathsf{id}} := \sigma_{\mathsf{id}}$, the message can be recovered as $\mathsf{ct}_2 - (\mathsf{ct}_1 \circ \mathsf{sk}_{\mathsf{id}})$.

Extension to the batched setting using an accumulator. We would like to extend the above basic BF-IBE construction to the batched setting. As mentioned earlier, simply concatenating the individual $\mathsf{sk}_{\mathsf{id}}$ values of all ids in the batch $S \subseteq \mathcal{I}$ doesn't lead to a succinct key for the batch. To remedy this, we use a cryptographic accumulator scheme. Such a scheme enables compressing a set $S = \{s_1, \ldots, s_n\}$ of items into a *succinct* public digest d. Using the set S and digest d, it is possible to compute a short cryptographic proof π_s proving that a specific element s is contained in S. The verification algorithm, given the digest d, claimed element s and proof π_s, outputs a bit indicating either accept or reject. The completeness of the scheme ensures that correctly generated proofs π_s for $s \in S$ always pass the verification check, whereas soundness ensures that it is hard for a computationally bounded adversary to compute valid proofs π_s for $s \notin S$.

Given such an accumulator scheme, a natural approach is to create a succinct digest d for the public batch of identities $S = \{\mathsf{id}_1, \ldots, \mathsf{id}_n\}$, and then compute a succinct secret key for the batch by setting sk_S to be a BLS signature on the digest d, i.e. $\mathsf{sk}_S := \sigma_S = \mathsf{msk} \cdot H(d) \in \mathbb{G}_1$. Now, one could hope to create an encryption scheme where a ciphertext ct, generated w.r.t. a specific identity id, is decryptable given a triple $(d, \pi_{\mathsf{id}}, \mathsf{sk}_S)$ as a witness, and if and only if the following two conditions hold: 1) π_{id} is a valid membership proof of id w.r.t. the digest d, 2) sk_S is a valid signature on the digest d.

Challenges and next steps. Although the above template conceptually works, it is not clear how to build an *efficient* encryption scheme satisfying the required properties and, in general, it seems to require a general purpose witness encryption [Gar+13; Gar+16; Tsa22; VWW22] which requires strong cryptographic assumptions and/or is often inefficient in practice.

We utilize the observation in [Gar+24] that the BF-IBE construction can be seen as a special case of a general technique which transforms a public linear constraint system, defined over some cryptographically hard, pairing-friendly groups, into an efficient witness encryption scheme. In such a constraint system, each constraint involves some public group elements (which are part of the statement) and some private group elements (which are part of the witness). The "linearity" condition requires that the witness group elements are never paired with each other in the constraint. To be specific, we can consider the following linear constraint system containing n constraints and m witness elements.

$$
\begin{aligned}
a_{1,1} \circ \boxed{w_1} + \ldots + a_{1,m} \circ \boxed{w_m} &= b_1 \\
a_{2,1} \circ \boxed{w_1} + \ldots + a_{2,m} \circ \boxed{w_m} &= b_2 \\
\ldots &= \ldots \\
a_{n,1} \circ \boxed{w_1} + \ldots + a_{n,m} \circ \boxed{w_m} &= b_n
\end{aligned}
$$

where $\{a_{i,j}\}_{i\in[n],j\in[m]}$ and $\{b_i\}_{i\in[n]}$ are public group elements in $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively whereas $\{w_i\}_{i\in[m]}$ are private witness elements in $\mathbb{G}_1$ (highlighted in grey). The above system of constraints can be succinctly expressed as

$$\mathbf{A} \circ \boxed{\mathbf{w}} = \mathbf{b}$$

where

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & \dots & a_{1,m} \\ \vdots & \vdots & \vdots \\ a_{n,1} & \dots & a_{n,m} \end{pmatrix}, \qquad \mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Given such a constraint system, a message $m \in \mathbb{G}_T$ can be witness-encrypted in the following way (analogous to BF-IBE encryption).

$$\mathsf{ct} = (\mathsf{ct}_\mathbf{1}, \mathsf{ct}_2) = (\mathbf{r}^T \cdot \mathbf{A}, \mathbf{r}^T \cdot \mathbf{b} + m)$$

where $\mathbf{r} \leftarrow \mathbb{Z}_p^n$ is a randomly sampled column vector and $\mathbf{r}^T$ denotes its transpose. Intuitively, one could think of the term $\mathbf{r}^T \cdot \mathbf{b}$ as a one-time-pad that is applied to the message m.

Given a witness $\mathbf{w}$, the message can be recovered from ct by simply computing $\mathsf{ct}_2 - \mathsf{ct}_1 \circ \mathbf{w}$ (again analogous to BF-IBE decryption).

Recall from our template discussed earlier that in the context of batched IBE, we want to create an encryption scheme where a ciphertext ct, generated w.r.t. a specific identity id, is decryptable given a witness triple $\mathbf{w} = (d, \pi_{\mathsf{id}}, \mathsf{sk}_S)$, iff the following two conditions hold: 1) π_{id} is a valid membership proof of id w.r.t. the digest d, 2) sk_S is a valid signature on the digest d. Each of these two conditions will induce a constraint on the witness $\mathbf{w}$. Therefore, in order for us to utilize the aforementioned general technique of building an encryption scheme from a constraint system, we need to select our cryptographic ingredients, namely the accumulator and signature scheme, in a careful way so that the induced constraints are linear w.r.t. the witness $\mathbf{w}$.

Observation 1. Our first obervation is that the well-known KZG commitment scheme [KZG10] satisfies the required property of linear verification. Let d denote the digest created out of a set $S \subseteq \mathcal{I} = \mathbb{Z}_p$. Without going too much into the details of the KZG scheme, we note that the digest d is created by interpolating a univariate polynomial f whose roots are all the elements in set S, and evaluating it at a secret point τ "in the exponent". In other words, d is simply $[f(\tau)]_1$. The verification of a membership proof $\pi_{\mathsf{id}} \in \mathbb{G}_1$ w.r.t. $\mathsf{id} \in \mathcal{I}$ involves checking the following constraint.

$$[1]_2 \circ d = ([\tau]_2 - [\mathsf{id}]_2) \circ \pi_{\mathsf{id}}$$

where $[\tau]_2$ is a public group element generated during a one-time setup phase.

Observation 2. Unfortunately, the BLS signature scheme [BLS01], which we have been discussing so far in the context of BF-IBE, does not satisfy the desired property of linear verification in our context. To be more specific, while the BLS verification constraint is linear w.r.t. a signature (which is the sk_S part of our witness $\mathbf{w}$), it is not linear w.r.t. the message being signed (which is the digest d part of our witness $\mathbf{w}$). In our context, the unmodified BLS verification constraint would look as follows,

$$[1]_2 \circ \mathsf{sk}_S = \mathsf{vk} \circ H(d)$$

As we can see, this constraint involves applying the hash function H on the digest d, which is a highly *non-linear* operation! It turns out that by adding a slight modification to the BLS signature scheme, we can restore linearity to the constraint. Let $\alpha \leftarrow \mathbb{G}_1$ be a random group element whose discrete logarithm is unknown to any party. Then, the modified BLS scheme would sign a message $m \in \mathbb{G}_1$ as $\sigma_m := \mathsf{msk} \cdot (m + \alpha)$. The verification constraint for a claimed signature σ on a message $m \in \mathbb{G}_1$ would simply check whether $[1]_2 \circ \sigma = \mathsf{vk} \circ (m + \alpha)$ holds. Translating the notations to our context, where we need to sign the digest d to produce sk_S, the verification constraint would be,

$$[1]_2 \circ \boxed{\mathsf{sk}_S} = \mathsf{vk} \circ (\boxed{d} + \alpha)$$

We note that this modification to the BLS scheme does not come for free. Firstly, it requires α to be generated as part of the setup. Secondly, if α is reused for signing more than once, then it can be used to forge signatures[5]. Thirdly, there is concrete forgery attack against this scheme in Type-1 and Type-2 pairing group (even when α is used only once)[6]. The first two limitations can be easily addressed by generating a fresh r as the output of a random oracle on a nonce (which could be the batch label in our context). To address the third limitation, we restrict ourselves to Type-3 pairing groups and prove the security of our overall scheme in the GGM.

Putting things together. Given these two ingredients (KZG commitments and modified BLS), we can now express our verification constraints in a form that is linear w.r.t. the witness $\mathbf{w} = (d, \pi_{\mathsf{id}}, \mathsf{sk}_S)$.

$$[1]_2 \circ \boxed{d} = ([\tau]_2 - [\mathsf{id}]_2) \circ \boxed{\pi_{\mathsf{id}}}$$
$$[1]_2 \circ \boxed{\mathsf{sk}_S} = \mathsf{vk} \circ (\boxed{d} + \alpha)$$

Rearranging the above constraint system, we get the following matrix form.

$$\underbrace{\begin{pmatrix} [1]_2 & [\mathsf{id}]_2 - [\tau]_2 & 0 \\ \mathsf{vk} & 0 & -[1]_2 \end{pmatrix}}_{\mathbf{A}} \circ \underbrace{\begin{pmatrix} \boxed{d} \\ \boxed{\pi_{\mathsf{id}}} \\ \boxed{\mathsf{sk}_S} \end{pmatrix}}_{\mathbf{w}} = \underbrace{\begin{pmatrix} [0]_T \\ -(\mathsf{vk} \circ \alpha) \end{pmatrix}}_{\mathbf{b}}$$

Given such a linear constraint system, a message m can be encrypted by sampling $\mathbf{r} \leftarrow \mathbb{Z}_p^2$ and computing

$$\mathsf{ct} = (\mathsf{ct}_{\mathbf{1}}, \mathsf{ct}_2) = (\mathbf{r}^T \cdot \mathbf{A}, \mathbf{r}^T \cdot \mathbf{b} + m)$$

as described earlier. Given a witness $\mathbf{w}$, the message can be recovered from ct by simply computing $\mathsf{ct}_2 - \mathsf{ct}_1 \circ \mathbf{w}$.

[5]Given signatures $\sigma_{m_1} := \mathsf{msk} \cdot (m_1 + \alpha)$ and $\sigma_{m_2} := \mathsf{msk} \cdot (m_2 + \alpha)$ on messages m_1 and m_2 respectively, one can easily get a valid signature $\sigma_{m_3} := 2\sigma_{m_1} - \sigma_{m_2} = \mathsf{msk} \cdot (2m_1 - m_2 + \alpha)$ on message $m_3 := (2m_1 - m_2)$

[6]In such groups, one can use $\mathsf{vk} = [\mathsf{msk}]_2$ to get $[\mathsf{msk}]_1 = \mathsf{msk} \cdot [1]_1$. Given a signature $\sigma_{m_1} := \mathsf{msk} \cdot (m_1 + \alpha)$ on message m_1, we can derive a signature $\sigma_{m_2} := \sigma_{m_1} + \mathsf{msk} \cdot [1]_1 = \mathsf{msk} \cdot (m_1 + 1 + \alpha)$ on message $m_2 = m_1 + 1$

3 Preliminaries

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

3.1 Bilinear Groups

We follow the notation used in [Gar+24], Section 3.1. A bilinear group, denoted as $\mathcal{BG}$, 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, denoted as e. This map takes one element from $\mathbb{G}_1$ and one element from $\mathbb{G}_2$ and produces an element in $\mathbb{G}_T$. The groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are referred to as source groups, while $\mathbb{G}_T$ is the target group. The groups $\mathbb{G}_1, \mathbb{G}_2$ have random generators g_1, g_2, and we use the notation $[x]_s$ to represent $x \cdot g_s$ in the group $\mathbb{G}_s$, for $s \in \{1, 2, T\}$, where $x \in \mathbb{Z}_p$ and the generator of $\mathbb{G}_T$ is $g_T = e(g_1, g_2)$. The group operation is additive, and therefore $[x]_s + [y]_s = [x + y]_s$.

We can represent the pairing operation $e([x]_1, [y]_2)$ as $[x]_1 \circ [y]_2 = [y]_2 \circ [x]_1 = [x \cdot y]_T$. (This notation makes it easier to write expressions which compose pairings with linear operations.) As a result, the operation $\circ$ is commutative and can be applied to vectors of equal length. For example, for $\mathbf{u} \in (\mathbb{G}_1)^n, \mathbf{v} \in (\mathbb{G}_2)^n$, where $\mathbf{u} = ([u_1]_1, \ldots, [u_n]_1)$ and $\mathbf{v} = ([v_1]_2, \ldots, [v_n]_2)$, we have that $\mathbf{u}^T \circ \mathbf{v} = [u_1 v_1 + \cdots u_n v_n]_T$. It is further possible to use this notation for matrix-vector multiplication. If $\mathbf{A} \in (\mathbb{G}_1)^{n \times m}$ and $\mathbf{b} \in (\mathbb{G}_2)^m$, then $\mathbf{A} \circ \mathbf{b}$ is the vector in $(\mathbb{G}_T)^n$ with the coordinates $(\mathbf{A}_1 \circ \mathbf{b}, \ldots, \mathbf{A}_n \circ \mathbf{b})$ where $\mathbf{A}_i$ is the i^{th} row vector of the matrix $\mathbf{A}$.

3.2 Generic group model (GGM)

At a high level, this model captures the class of 'generic' adversaries - adversaries that don't exploit concrete representations of the elements of the group and only perform generic group operations. This model is aimed to captures the possible *algebraic* attacks that an adversary can perform. The following description of Shoup's GGM [Sho97] is taken from [Zha22]. Let $p \in \mathbb{Z}$ be a positive integer, and let $S \subseteq \{0,1\}^*$ be a set of strings of cardinality at least p. We will assume an upper bound is known on the length of strings in S. An arbitrary group $\mathbb{G}$ of prime order p is generically modeled by sampling a *random* injection $L : \mathbb{Z}_p \to S$, which we will call the *labeling function*. We will think of $L(x)$ as corresponding to (the string representation) of g^x, where $g \in \mathbb{G}$ is a fixed generator of the group $\mathbb{G}$. All parties are able to make the following queries:

- **Labeling queries.** The party submits $x \in \mathbb{Z}_p$, and receives $L(x)$.
- **Group operations.** The party submits $(\ell_1, \ell_2, a_1, a_2) \in S^2 \times \mathbb{Z}_p^2$. If there exists $x_1, x_2 \in \mathbb{Z}_p$ such that $L(x_1) = \ell_1$ and $L(x_2) = \ell_2$, then the party receives $L(a_1 x_1 + a_2 x_2)$. Otherwise, the party receives $\perp$.

We note that the labeling map L need not be explicitly materialized all at once. In fact, it is typical in security proofs to sample such a map in a lazy fashion as the queries are made. When

performing such a lazy sampling, we will think of L as a "dictionary" and use $x \in L$ to mean the action of checking whether the element x exists as a "key" in L.

In a bilinear group $\mathcal{BG}$, the parties are equipped with labeling queries and group operations in $\mathbb{G}_i$, with a random labeling function $L_i : \mathbb{Z}_p \to S_i$ for $S_i \subseteq \{0,1\}^*$ and $i \in \{1, 2, T\}$. Additionally, the parties can make the following query for pairing operation [BBG05].

- **Pairing operation.** The party submits $(\ell_1, \ell_2) \in S_1 \times S_2$. If there exists $x_1, x_2 \in \mathbb{Z}_p$ such that $L_1(x_1) = \ell_1$ and $L_2(x_2) = \ell_2$, then the party receives $L_T(x_1 \cdot x_2)$. Otherwise, the party receives $\perp$.

The typical GGM does not capture an adversary's ability to "hash" arbitrary strings into the group To model this, we can extend the GGM with an appropriate hashing oracle as discussed in [LPS23; Bau+23]. Here, a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ is sampled at random and parties can make the following query.

- **Hash query.** The party submits $x \in \{0,1\}^*$ and receives $L(H(x))$.

Security proofs in the GGM. The security of a cryptographic scheme is usually defined by a game where two parties, namely the challenger and the adversary, interact as per some specification and the challenger finally outputs a bit b indicating the verdict of the game ($b = 1$ for win and $b = 0$ for lose). When writing security proofs in the GGM, the challenger is supposed to simulate the GGM oracle towards the adversary. Let $n \in \mathbb{Z}$ be a positive integer. The game usually involves the challenger sampling some secrets $x_1, \ldots, x_n \in \mathbb{Z}_p$ and leaking $L(f(x_1, \ldots, x_n))$ to the adversary for some f.

A standard technique for security proofs in the GGM, introduced in [Sho97], is the following. Instead of explicitly sampling secret $x_i \in \mathbb{Z}_p$, the challenger replaces it with an indeterminate X_i. Additionally, the domain of the labeling function is expanded from $\mathbb{Z}_p$ to $\mathbb{Z}_p[X_1, \ldots, X_n]$, the set of all n variate polynomials with coefficients in $\mathbb{Z}_p$. Now, instead of leaking $L(f(x_1, \ldots, x_n))$ to the adversary, the challenger leaks $L(f(X_1, \ldots, X_n))$. This model, where the challenger substitutes all the secrets with their corresponding indeterminates, is known as the "symbolic model". Proving security in this model boils down to showing independence between a target polynomial and the polynomials whose labels/encodings are present in the view of the adversary. We will use this technique in proving the security of our scheme.

3.3 Polynomial Commitment

A polynomial commitment enables the computation of a compact value, denoted as *com*, for a polynomial f that may have a high degree over a finite field $\mathbb{F}$. Subsequently, one can compute concise openings to prove that the polynomial committed to by *com* evaluates to some value β at a specific point α. The polynomial commitment must be binding, meaning it should be infeasible to open the same point to two distinct values. We refer the reader to [KZG10] for the detailed definition of polynomial commitments.

The KZG polynomial commitment scheme [KZG10] uses as public parameters powers of a secret point τ in the exponent of a group generator. In the group $\mathbb{G}_1$ used in our construction these would be the values $[\tau]_1, \ldots, [\tau^d]_1$, where d is an upper bound on the degree of the polynomial.

These parameters are computed in a setup phase, and the value of τ is kept secret. Committing to a polynomial is done by evaluating it in the exponent at point τ, namely computing $[f(\tau)]_1$. A crucial property is that this computation can be done using the public powers of τ, but without knowledge of τ itself.

Theorem 3.1 ([KZG10; Chi+20]). *If the d-DLOG assumption holds with respect to parameter generation algorithm of the KZG commitment scheme described in [KZG10], then that commitment scheme is a correct and binding polynomial commitment scheme in the AGM, according to the definitions of [KZG10; Chi+20].*

4 Defining Selective Batched Identity Based Encryption

4.1 Syntax

As in a standard IBE [BF01], a Selective Batched IBE will include a Setup algorithm (with an additional parameter for batch size) which generates public parameters and KeyGen algorithm that generates a public key pk and a master secret key msk. Typically, KeyGen will be executed by a central authority which privately stores msk and publishes pk publicly. Using the pk, anyone can encrypt a message m w.r.t. a specific id, along with a batch label t, to produce a ciphertext. In contrast to a standard IBE where the authority issues identity specific secret keys, our Selective Batched IBE will enables secret key issuance for a *batch* of identities. To capture this, we split the secret key derivation process into two parts: 1) A Digest algorithm that, given a batch of identities, produces a digest, 2) A ComputeKey algorithm that, given a digest and the batch label, produces a secret key sk. We note that only the ComputeKey algorithm uses the master secret key. The Digest algorithm only uses public information that is accessible to all participants. Finally, anyone holding the digest-batch label-specific key secret key sk can use the Decrypt algorithm to decrypt all ciphertexts whose identities were part of the digest.

Definition 4.1 (Selective Batched IBE Syntax)**.** A Selective Batched IBE scheme SBIBE is specified by six algorithms: Setup, KeyGen, Encrypt, Decrypt, Digest, ComputeKey.

- $\mathsf{Setup}(1^\lambda, 1^B) \to \mathsf{params}$: A randomized algorithm that takes as input a security parameter $\lambda \in \mathbb{N}$ and a batch size $B = B(\lambda)$. It outputs params (system parameters) which includes a description of the message space $\mathcal{M}$, identity space $\mathcal{I}$, batch label space $\mathcal{T}$ and ciphertext space $\mathcal{C}$.
- $\mathsf{KeyGen}(\mathsf{params}) \to (\mathsf{msk}, \mathsf{pk})$: a randomized algorithm that takes as input params and outputs msk (master secret key) and pk (public key).
- $\mathsf{Encrypt}(\mathsf{pk}, m, \mathsf{id}, t) \to c$: a randomized algorithm that takes as input a message $m \in \mathcal{M}$, an identity $\mathsf{id} \in \mathcal{I}$, a batch label $t \in \mathcal{T}$, public key pk and outputs a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}) \to d$: a deterministic algorithm that takes as input the public key pk and a list of identities $\mathsf{id}_1, \ldots, \mathsf{id}_B$ where each $\mathsf{id}_i \in \mathcal{I}$. It outputs a digest d.
- $\mathsf{ComputeKey}(\mathsf{msk}, d, t) \to \mathsf{sk}$: a deterministic algorithm that takes as input the master secret key msk, digest d, batch label t and outputs a digest-batch label-specific secret key sk.

- $\mathsf{Decrypt}(c, \mathsf{sk}, d, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}, \mathsf{id}, t) \rightarrow m$: a deterministic algorithm that takes as input a ciphertext c, secret key sk, digest d, a list of identities $\mathsf{id}_1, \ldots, \mathsf{id}_B$ and an identity-batch label pair (id, t). It outputs a message $m \in \mathcal{M}$.

Remark. We note that the syntax doesn't enforce any restrictions on how the batch list $\{\mathsf{id}_1, \ldots, \mathsf{id}_B\}$ is selected as input for the Digest algorithm. We leave this choice to higher-level applications that use the SBIBE primitive. One such application, namely mempool privacy, is discussed in Section 6.4. Other applications are discussed in Section 1.2.

4.2 Correctness, Non-triviality and Security

The above algorithms should satisfy the following requirements.

Definition 4.2 (Selective Batched IBE Correctness)**.** For all $\lambda \in \mathbb{N}, B \in \mathbb{N}, m \in \mathcal{M}, t \in \mathcal{T}, \mathsf{id} \in \mathcal{I}, S \subseteq \mathcal{I}$ s.t. $|S| = B$ and $\mathsf{id} \in S$, the following should hold:

$$\Pr\left[\mathsf{Decrypt}(c, \mathsf{sk}, d, S, \mathsf{id}, t) = m \;\middle|\; \begin{array}{c} \mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^B) \\ (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{params}) \\ c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m, \mathsf{id}, t) \\ d \leftarrow \mathsf{Digest}(\mathsf{pk}, S) \\ \mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, d, t) \end{array}\right] = 1$$

Definition 4.3 (Selective Batched IBE Non-triviality/Efficiency)**.** We require that the running time of $\mathsf{ComputeKey}$ be independent of the batch size B (which implies that the digest d and sk are also independent of B)

Remark. We enforce the above requirement for the following reasons: 1) Without this requirement, one could come up with a trivial scheme using standard IBE where the secret key sk for a set of ids $\{\mathsf{id}_1, \ldots, \mathsf{id}_B\}$ and batch label t is simply the standard IBE secret keys for identities $\{\mathsf{id}_1||t, \ldots, \mathsf{id}_B||t\}$. Here, the running time of $\mathsf{ComputeKey}$ and its output sk will be $O(B)$. 2) This feature, while already useful in a non-threshold setting, becomes even more useful in a threshold setting where the msk is split among multiple authority members (using a secret sharing scheme) and they securely emulate the execution of $\mathsf{ComputeKey}$ procedure using their share of msk to produce sk. In such a setting, the above requirement will ensure that the running time and communication cost of the secure emulation is independent of the batch size (which can be a huge cost saving in practice). We refer the readers to Appendix A for more details regarding the threshold setting.

Our definition of security is an adaptation of the standard definition of IBE by Boneh et. al. [BF01] and captures the fact that any ciphertext c created w.r.t. an identity id^* and batch label t^* remains hidden as long as the id^* is not included in any of the batches for which sk has been released. Moreover, we allow each batch label to be used only once as this is what our construction achieves and suffices for many of the discussed applications (where batch label can be considered as a "round number")

Definition 4.4 (Selective Batched Identity Based Encryption Security)**.** We define a security game $\mathsf{Expt}^{\mathsf{SBIBE}}_{\mathcal{A},b}(1^\lambda, B)$ with respect to adversary $\mathcal{A}$ in the box below.

We say that a batched IBE scheme is secure if for all $B \in \mathbb{N}$, for all PPT adversaries $\mathcal{A}$ there exists some negligible function ϵ_A such that the following holds:

$$\left|\Pr[\mathsf{Expt}^{\mathsf{SBIBE}}_{\mathcal{A},0}(1^\lambda, B) = 1] - \Pr[\mathsf{Expt}^{\mathsf{SBIBE}}_{\mathcal{A},1}(1^\lambda, B) = 1]\right| < \epsilon_A(\lambda).$$

The security game $\mathsf{Expt}^{\mathsf{SBIBE}}_{\mathcal{A},b}(1^\lambda, B)$.

Setup: The challenger takes as input the security parameter λ and the batch size B. It runs $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^B)$, and then runs $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$. Finally, it sends $(\mathsf{params}, \mathsf{pk})$ to $\mathcal{A}$.

The rest of the game proceeds in rounds, as follows.

Pre-challenge queries: $\mathcal{A}$ may issue an arbitrary number of *key computation queries:*

- A sends a list ids of B identities along with a batch label t to the challenger.
- If a key computation query has already been made with batch label t, the challenger halts the game.
- Otherwise, the challenger does the following:
  - Compute a digest $d \leftarrow \mathsf{Digest}(\mathsf{pk}, \mathsf{ids})$ of the ids in ids using public key pk.
  - Compute a secret key $\mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, d, t)$, using the digest d computed from the previous step.
  - Send sk to $\mathcal{A}$.

Challenge round: Once during the game, $\mathcal{A}$ may decide that the current round is the *challenge round.* The challenge round proceeds as follows:

- $\mathcal{A}$ sends two messages $m_0, m_1 \in \mathcal{M}$ and an identity-batch label pair (id^*, t^*) on which it wishes to be challenged.
- If key computation query (ids, t) has already been made with batch label $t = t^*$ and where $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.
- Otherwise, the challenger computes $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b, \mathsf{id}^*, t^*)$ and sends c to $\mathcal{A}$.

Post-challenge queries: After the challenger round, $\mathcal{A}$ may again issue an arbitrary number of *key computation queries,* with the additional restriction that A cannot query (t^*, ids) with $\mathsf{id}^* \in \mathsf{ids}$:

- $\mathcal{A}$ sends a list ids of B identities along with a batch label t to the challenger.

- If a key computation query has already been made with batch label t **or if** $t = t^*$ **and** $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.
- Otherwise, the challenger does the following:
  - Compute a digest $d \leftarrow \mathsf{Digest}(\mathsf{pk}, \mathsf{ids})$ of the ids in ids using public key pk.
  - Compute a secret key $\mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, d, t)$, using the digest d computed from the previous step.
  - Send sk to $\mathcal{A}$.

Output: At any point in time, $\mathcal{A}$ can decide to halt and output a bit $b' \in \{0, 1\}$. The game then halts with the same output b'.

5 Our Selective Batched Identity Based Encryption construction

5.1 Construction

In this section, we will provide a construction of SBIBE scheme based on Kate et. al. [KZG10] polynomial commitment scheme and the modified BLS signature scheme that we discussed in Section 2. We show the formal construction in Fig. 1. The key details of the construction are as follows. The construction is in a Type-3 asymmetric pairing setting, with groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order p, and a pairing operation $e(\mathbb{G}_1, \mathbb{G}_2) \to \mathbb{G}_T$. The message space is $\mathbb{G}_T$ and the identity space is $\mathbb{Z}_p$. The construction also uses a hash function H whose range is $\mathbb{G}_1$.

Key generation includes choosing a random master secret key $\mathsf{msk} \leftarrow \mathbb{Z}_p$ and a random secret parameter $\tau \leftarrow \mathbb{Z}_p$. The secret key is msk, while the public key includes a powers-of-Tau setting of order B in $\mathbb{G}_1$, namely $([\tau]_1, \ldots, [\tau^B]_1$, as well as the following two elements in $\mathbb{G}_2$, $[\tau]_2$ and $[\mathsf{msk}]_2$, corresponding to raising the generator $g_2 \in \mathbb{G}_2$ to the powers of τ and msk.

The encryption of a message m with identity id to a batch label t, includes computing a matrix $\mathbf{A} \in (\mathbb{G}_2)^{2\times 3}$ and a vector $\mathbf{b} \in (\mathbb{G}_T)^2$. In addition to being dependent on the public key, $\mathbf{A}$ depends on the identity id, whereas $\mathbf{b}$ depends on the batch label (and therefore $\mathbf{b}$ is identical for all messages encrypted to this batch label). The encryption is $c = (c_1, c_2) = (\mathbf{r}^T \cdot \mathbf{A}, \mathbf{r}^T \cdot \mathbf{b} + m)$, where $\mathbf{r}$ is a random column vector in $(\mathbb{Z}_p)^2$. Note that c_1 is a vector of dimension 3 in $\mathbb{G}_2$, and c_2 is an element of $\mathbb{G}_T$.

The digest algorithm takes B identities and then interpolates a polynomial of degree B in $\mathbb{Z}_p$ whose roots are these identities, and whose leading coefficient is 1. The digest d is the KZG commitment of this polynomial, namely the value of this polynomial at the point $[\tau]_1$.

The compute-key algorithm outputs the secret key $\mathsf{sk} := \mathsf{msk} \cdot (d + H(t)) \in \mathbb{G}_1$. This is the only algorithm that uses the master secret key msk.

Decryption is computed independently for every ciphertext c. The decryption of a message with identity id involves interpolating a polynomial which has roots in all identities in the digest, *except* for id, and computing a KZG opening proof π using this polynomial. The ciphertext is parsed as (c_1, c_2), and decrypted by computing message $c_2 - c_1 \circ (d, \pi, \mathsf{sk})^T$.

Theorem 5.1. *Assuming Type-3 pairing group $\mathcal{BG}$, there exists a construction (Fig. 1) for Selective Batched IBE which is secure in the Generic group model.*

SBIBE **construction**

- $\mathsf{Setup}(1^\lambda, 1^B)$: Output three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order p, where p is a λ-bit prime, equipped with generators g_1, g_2, g_T, respectively, and an efficiently computable pairing operation $\circ : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$. Set the message space $\mathcal{M} := \mathbb{G}_T$, identity space $\mathcal{I} := \{0, \ldots, p-1\}$, and batch label space $\mathcal{T} := \{0,1\}^\lambda$. Also output a randomly sampled hash function $H : \mathcal{T} \to \mathbb{G}_1$.
- $\mathsf{KeyGen}(\mathsf{params})$: Sample $\mathsf{msk} \leftarrow \mathbb{Z}_p$ and $\tau \leftarrow \mathbb{Z}_p$. Output $\mathsf{msk}, \mathsf{pk} := ([\tau]_1, \ldots, [\tau^B]_1, [\tau]_2, [\mathsf{msk}]_2)$.
- $\mathsf{Encrypt}(\mathsf{pk}, m, \mathsf{id}, t)$: Let $\mathbf{A}$ be a matrix in $(\mathbb{G}_2)^{2\times 3}$ and $\mathbf{b}$ be a vector in $(\mathbb{G}_T)^2$, defined as follows.

$$\mathbf{A} := \begin{pmatrix} [1]_2 & [\mathsf{id}]_2 - [\tau]_2 & 0 \\ [\mathsf{msk}]_2 & 0 & -[1]_2 \end{pmatrix}$$

$$\mathbf{b} := \begin{pmatrix} [0]_T \\ -([\mathsf{msk}]_2 \circ H(t)) \end{pmatrix}$$

  Sample a (column) vector $\mathbf{r} = (r_1, r_2) \leftarrow (\mathbb{Z}_p)^2$ and output the ciphertext c where

$$c = (\mathbf{r}^T \cdot \mathbf{A}, \mathbf{r}^T \cdot \mathbf{b} + m)$$

- $\mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\})$: Let $f(X) = \sum_{i=0}^{B} f_i \cdot X^i$ be a univariate polynomial of degree B over $\mathbb{Z}_p$ with roots at $\mathsf{id}_1, \ldots, \mathsf{id}_B$ and leading coefficient 1. Output digest $d := \sum_{i=0}^{B} f_i \cdot [\tau^i]_1$.
- $\mathsf{ComputeKey}(\mathsf{msk}, d, t)$: Output the secret key $\mathsf{sk} := \mathsf{msk} \cdot (d + H(t))$.
- $\mathsf{Decrypt}(c, \mathsf{sk}, d, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}, \mathsf{id}, t)$: Let $q(X) = \sum_{i=0}^{B-1} q_i \cdot X^i$ be a univariate polynomial of degree $B-1$ with roots at $\{\mathsf{id}_1, \ldots, \mathsf{id}_B\} \setminus \{\mathsf{id}\}$ and leading coefficient 1. Set $\pi := \sum_{i=0}^{B-1} q_i \cdot [\tau^i]_1$ and set $\mathbf{w}$ to be the following vector.

$$\mathbf{w} = \begin{pmatrix} d \\ \pi \\ \mathsf{sk} \end{pmatrix}$$

  Finally, parse c as (c_1, c_2) and output the decrypted message $m^* := c_2 - c_1 \circ \mathbf{w}$.

Figure 1: Our construction for Selective Batched Identity Based Encryption

5.2 Analysis

5.2.1 Efficiency

We discuss the computation cost of each of the algorithms in Fig. 1. Setup requires $O(\lambda)$ operations. KeyGen requires $O(B)$ group exponentiations. Encrypt requires $O(1)$ group exponentiations. Digest and Decrypt require $O(B \log B)$ field multiplications (via DFT) and $O(B)$ group exponentiations. ComputeKey requires $O(1)$ group operations (independent of batch size B). We summarize the concrete parameter sizes below in Table 1.

Parameter	Size
Public parameters size	$B\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$
Ciphertext size	$3\|\mathbb{G}_2\| + \|\mathbb{G}_T\|$
Digest size	$\|\mathbb{G}_1\|$
Decryption key size	$\|\mathbb{G}_1\|$

Table 1: Parameter sizes for our scheme.

5.2.2 Correctness

The correctness of the scheme is straight forward. Given a ciphertext $c = (c_1, c_2)$, and a witness $\mathbf{w}$ generated as per the specified algorithms, we have the following decrypted message.

$$
\begin{aligned}
m^* &= c_2 - c_1 \circ \mathbf{w} \\
&= \mathbf{r}^T \cdot \mathbf{b} + m - (\mathbf{r}^{\mathbf{T}} \cdot \mathbf{A}) \circ \mathbf{w} \\
&= -r_2([\mathsf{msk}]_2 \circ H(t)) + m - \Big([r_1 + r_2 \cdot \mathsf{msk}]_2, [r_1 \cdot \mathsf{id}]_2 - [r_1 \cdot \tau]_2, -[r_2]_2\Big) \circ \begin{pmatrix} d \\ \pi \\ \mathsf{sk} \end{pmatrix} \\
&= -r_2([\mathsf{msk}]_2 \circ H(t)) + m - \Big([r_1 + r_2 \cdot \mathsf{msk}]_2, [r_1 \cdot \mathsf{id}]_2 - [r_1 \cdot \tau]_2, -[r_2]_2\Big) \\
&\circ \begin{pmatrix} [\Pi_{i=1}^{B}(\tau - \mathsf{id}_i)]_1 \\ [\Pi_{i=1, \mathsf{id}_i \neq \mathsf{id}}^{B}(\tau - \mathsf{id}_i)]_1 \\ \mathsf{msk} \cdot ([\Pi_{i=1}^{B}(\tau - \mathsf{id}_i)]_1 + H(t)) \end{pmatrix} \\
&= -r_2([\mathsf{msk}]_2 \circ H(t)) + m - [r_1 \cdot \Pi_{i=1}^{B}(\tau - \mathsf{id}_i) + r_2 \cdot \mathsf{msk} \cdot \Pi_{i=1}^{B}(\tau - \mathsf{id}_i) \\
&- r_1 \cdot \Pi_{i=1}^{B}(\tau - \mathsf{id}_i) - r_2 \cdot \mathsf{msk} \cdot \Pi_{i=1}^{B}(\tau - \mathsf{id}_i)]_T + [r_2]_2 \circ \mathsf{msk} \cdot H(t) \\
&= m
\end{aligned}
$$

5.2.3 Security

We will prove the security of our scheme in the GGM model equipped with hash queries. In this model, the challenger will implement the group oracle and the hash oracle (along with an oracle for key computation queries as defined in the security game $\mathsf{Expt}^{\mathsf{SBIBE}}$ earlier).

Theorem 5.2. *For all $B \in \mathbb{N}$ and all unbounded adversaries $\mathcal{A}$ making at most q queries (including queries to the group oracle, hash oracle, and key computation queries), we have:*

$$\left|\Pr[\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}_{\mathcal{A},0}(1^\lambda, B) = 1] - \Pr[\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}_{\mathcal{A},1}(1^\lambda, B) = 1]\right| \leq 2\binom{q+B+5}{2}\frac{(B+2)}{p}$$

where $\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}$ *refers to the same experiment* $\mathsf{Expt}^{\mathsf{SBIBE}}$ *as defined in Definition 4.4 except that we specialize it for our specific Construction (Fig. 1) and model it in the GGM, i.e., all the group and hash operations performed by the adversary are simulated by the challenger as defined in the GGM model (Section 3.2). We formally define the experiment below.*

The security game $\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}_{\mathcal{A},b}(1^\lambda, B)$.

Setup: The challenger takes as input the security parameter λ and the batch size B. It runs $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^B)$. Let $S_1 = S_2 = S_T = \{0,1\}^{p'}$ where $2^{p'} \geq p$ and p is a prime (denoting the group order) contained in params. Then it initializes the maps including the labeling function $L_i : \mathbb{Z}_p \to S$ for each $i \in \{1, 2, T\}$ and the hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. During the KeyGen phase, it performs the following steps:

- Sample $e_{\mathsf{msk}} \leftarrow S_2$, $\mathsf{msk} \leftarrow \mathbb{Z}_p$. Set $L_2(\mathsf{msk}) := e_{\mathsf{msk}}$ and $S_2 := S_2 \setminus \{e_{\mathsf{msk}}\}$.
- Sample $\tau \leftarrow \mathbb{Z}_p$.
- For $i \in [B-1]$, sample $e_{\tau^i} \leftarrow S_1$ and set $L_1(\tau^i) := e_{\tau^i}$ and $S_1 := S_1 \setminus \{e_{\tau^i}\}$.
- Sample $e' \leftarrow S_2$ and set $L_2(\tau) = e'$ and $S_2 := S_2 \setminus \{e'\}$.

It sets $\mathsf{pk} = (L_1(\tau), \ldots, L_1(\tau^B), L_2(\tau), L_2(\mathsf{msk}))$ and sends $(\mathsf{params}, \mathsf{pk})$ to $\mathcal{A}$.

The rest of the game proceeds in rounds, as follows.

Pre-challenge queries: $\mathcal{A}$ may issue an arbitrary number of the following types of queries:

Labeling query: For each labeling query for group $\mathbb{G}_i$, the challenger receives a value $v \in \mathbb{Z}_p$ from $\mathcal{A}$. If $v \notin L_i$, it samples $z \leftarrow S_i$, sets $L_i(v) := z$ and $S_i := S_i \setminus \{z\}$. It sends $L_i(v)$ to $\mathcal{A}$.

Group operation: For each group operation query for group $\mathbb{G}_i$, the challenger receives $(\ell_1, \ell_2, a_1, a_2) \in (\{0,1\}^{p'})^2 \times \mathbb{Z}_p^2$. If there doesn't exists $x_1, x_2 \in \mathbb{Z}_p$ such that $L_i(x_1) = \ell_1$ and $L_i(x_2) = \ell_2$, then send $\perp$ to $\mathcal{A}$. Otherwise, execute an internal labeling query step on group $\mathbb{G}_i$ with input $x_3 := a_1 x_1 + a_2 x_2$ and send $L_i(x_3)$ to $\mathcal{A}$.

Pairing operation: The challenger receives $(\ell_1, \ell_2) \in \{0,1\}^{p'} \times \{0,1\}^{p'}$. If there doesn't exist $x_1, x_2 \in \mathbb{Z}_p$ such that $L_1(x_1) = \ell_1$ and $L_2(x_2) = \ell_2$, then the challenger sends $\perp$ to $\mathcal{A}$. Otherwise, it computes $x_3 = x_1 \cdot x_2$. It executes an internal labeling query step for group $\mathbb{G}_T$ with input x_3 and sends $L_T(x_3)$ to $\mathcal{A}$.

Hash query: For each hash query, the challenger receives a string $s \in \{0,1\}^*$ from $\mathcal{A}$. If $s \notin H$, the challenger samples $z \leftarrow \mathbb{Z}_p$. It sets $H(s) := z$ and executes an internal labeling query step for group $\mathbb{G}_1$ with input z. It sends $L_1(z)$ to $\mathcal{A}$.

Key computation query:

- $\mathcal{A}$ sends a list ids of B identities, $S = \{\mathsf{id}_1, \ldots, \mathsf{id}_B\} \subseteq \mathcal{I}$, along with a batch label $t \in \mathcal{T}$ to the challenger.
- If a key computation query has already been made with batch label t, the challenger halts the game.
- Otherwise, the challenger does the following:
  - Let $f(X) = \sum_{i=0}^{B} f_i \cdot X^i$ be a univariate polynomial of degree B over $\mathbb{Z}_p$ with roots at $\mathsf{id}_1, \ldots, \mathsf{id}_B$ and leading coefficient 1. If $f(\tau) \notin L_1$, then execute an internal labeling query step for group $\mathbb{G}_1$ with input $f(\tau)$.
  - If $t \notin H$, execute an internal hash query step with input t.
  - Let $P := (f(\tau) + H(t)) \cdot \mathsf{msk}$. If $P \notin L_1$, then execute an internal labeling query step for group $\mathbb{G}_1$ with input P.
  - Finally, send the secret key $\mathsf{sk} := L_1(P)$ to $\mathcal{A}$.

Challenge round: Once during the game, $\mathcal{A}$ may decide that the current round is the *challenge round.* The challenge round proceeds as follows:

- $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^{p'}$ and an identity-batch label pair (id^*, t^*) on which it wishes to be challenged.
- If key computation query (ids, t) has already been made with batch label $t = t^*$ and where $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.
- Otherwise, the challenger executes the following steps.
  - If $t^* \notin H$, execute a hash query step internally using input t^*.
  - Let $\mathbf{A}$ be a matrix and $\mathbf{b}$ be a vector defined as follows.

$$\mathbf{A} := \begin{pmatrix} 1 & \mathsf{id}^* - \tau & 0 \\ \mathsf{msk} & 0 & -1 \end{pmatrix}$$

$$\mathbf{b} := \begin{pmatrix} 0 \\ -H(t^*) \cdot \mathsf{msk} \end{pmatrix}$$

  - Sample a (column) vector $\mathbf{r} \leftarrow (\mathbb{Z}_p^*)^2$. Compute a list $\mathbf{y}$ of four $\mathbb{Z}_p$ values where

$$\mathbf{y} = (\mathbf{r}^T \cdot \mathbf{A}, \mathbf{r}^T \cdot \mathbf{b})$$

– Parse $\mathbf{y}$ as (y_0, y_1, y_2, y_3). For each $i \in \{0,1,2,3\}$, execute an internal labeling query step for group $\mathbb{G}_2$ with input y_i.
– Send the ciphertext $\mathbf{c} = (L_2(y_0), L_2(y_1), L_2(y_2), L_2(y_3 + L_2^{-1}(m_b)))$ to $\mathcal{A}$

Post-challenge queries: After the challenger round, $\mathcal{A}$ may again issue an arbitrary number of all the query types that were part of Pre-challenge queries with the following additional restriction on *key computation queries*: $\mathcal{A}$ cannot query (t^*, ids) with $\mathsf{id}^* \in \mathsf{ids}$. More formally, we have the following:

Key computation query:

- $\mathcal{A}$ sends a list ids of B identities along with a batch label t to the challenger.
- If a key computation query has already been made with batch label t **or if** $t = t^*$ **and** $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.
- Otherwise, the challenger executes the same steps as described in the Pre-challenge queries.

Output: At any point in time, $\mathcal{A}$ can decide to halt and output a bit $b' \in \{0,1\}$. The game then halts with the same output b'.

Proof. To prove Thm. 5.2, we will proceed in two steps. In the first step, we will show that the difference between symbolic versions of the experiments $\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{SBIBE,GGM}}$ and $\mathsf{Expt}_{\mathcal{A},1}^{\mathsf{SBIBE,GGM}}$, denoted by $\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{SBIBE,SM}}$ and $\mathsf{Expt}_{\mathcal{A},1}^{\mathsf{SBIBE,SM}}$ respectively, is zero (Lemma 5.3). In the second step, we will show that for $b \in \{0,1\}$, the difference between $\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE,GGM}}$ and $\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE,SM}}$ is at most $\epsilon = \binom{q+B+5}{2}\frac{(B+2)}{p}$ (Lemma 5.4). Combining these two steps implies the following:

$$\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{SBIBE,GGM}} \approx_\epsilon \mathsf{Expt}_{\mathcal{A},0}^{\mathsf{SBIBE,SM}} \equiv \mathsf{Expt}_{\mathcal{A},1}^{\mathsf{SBIBE,SM}} \approx_\epsilon \mathsf{Expt}_{\mathcal{A},1}^{\mathsf{SBIBE,GGM}}$$

which implies that $\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{SBIBE,GGM}} \approx_{2\epsilon} \mathsf{Expt}_{\mathcal{A},1}^{\mathsf{SBIBE,GGM}}$ and completes the proof of Thm. 5.2. □

Corollary 5.2.1. *For $p = O(2^\lambda)$, $B = \mathsf{poly}(\lambda)$ and $q = \mathsf{poly}(\lambda)$ and all unbounded adversaries $\mathcal{A}$ making at most q queries (including queries to the group oracle, hash oracle, and key computation queries), we have:*

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{SBIBE,GGM}}(1^\lambda, B) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},1}^{\mathsf{SBIBE,GGM}}(1^\lambda, B) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Lemma 5.3. *For all $B \in \mathbb{N}$ and all unbounded adversaries $\mathcal{A}$, we have:*

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{SBIBE,SM}}(1^\lambda, B) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},1}^{\mathsf{SBIBE,SM}}(1^\lambda, B) = 1]\right| = 0$$

The security game $\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE,SM}}(1^\lambda, B)$.

Setup: The challenger takes as input the security parameter λ and the batch size B. It runs $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^B)$. Let $S_1 = S_2 = S_T = \{0,1\}^{p'}$ where $2^{p'} \geq p$ and p is a prime (denot-

ing the group order) contained in params. Then it initializes the maps including the labeling function $L_i : \mathbb{Z}_p \to S$ for each $i \in \{1, 2, T\}$ and the hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. During the KeyGen phase, it performs the following steps:

- Sample $e_{\mathsf{msk}} \leftarrow S_2$, sets $L_2(X_{\mathsf{msk}}) := e_{\mathsf{msk}}$ where X_{msk} is an indeterminate corresponding to the master secret key msk. Set $S_2 := S_2 \setminus \{e_{\mathsf{msk}}\}$.
- Let X_τ be an indeterminate.
- For $i \in [B-1]$, sample $e_{\tau^i} \leftarrow S_1$ and set $L_1(X_\tau^i) := e_{\tau^i}$ and $S_1 := S_1 \setminus \{e_{\tau^i}\}$.
- Also, sample $e' \leftarrow S_2$ and set $L_2(X_\tau) := e'$ and $S_2 := S_2 \setminus \{e'\}$.

It sets $\mathsf{pk} = (L_1(X_\tau), \ldots, L_1(X_\tau^B), L_2(X_\tau), L_2(X_{\mathsf{msk}}))$ and sends $(\mathsf{params}, \mathsf{pk})$ to $\mathcal{A}$.

The rest of the game proceeds in rounds, as follows.

Pre-challenge queries: $\mathcal{A}$ may issue an arbitrary number of the following types of queries:

Labeling query: For each labeling query for group $\mathbb{G}_i$, the challenger receives a value $v \in \mathbb{Z}_p$ from $\mathcal{A}$. If $v \notin L_i$, it samples $z \leftarrow S_i$, sets $L_i(v) := z$ and $S_i := S_i \setminus \{z\}$. It sends $L_i(v)$ to $\mathcal{A}$.

Group operation: For each group operation query for group $\mathbb{G}_i$, the challenger receives $(\ell_1, \ell_2, a_1, a_2) \in \{0,1\}^{{p'}^2} \times \mathbb{Z}_p^2$. If there doesn't exist polynomials $x_1, x_2 \in \mathbb{Z}_p[*]$ such that $L(x_1) = \ell_1$ and $L(x_2) = \ell_2$, then send $\perp$ to $\mathcal{A}$. Otherwise, execute an internal labeling query step for group $\mathbb{G}_i$ with polynomial $x_3 = a_1 x_1 + a_2 x_2$ and send $L_i(x_3)$ to $\mathcal{A}$.

Pairing operation: The challenger receives $(\ell_1, \ell_2) \in \{0,1\}^{p'} \times \{0,1\}^{p'}$. If there doesn't exist polynomials $x_1, x_2 \in \mathbb{Z}_p[*]$ such that $L_1(x_1) = \ell_1$ and $L_2(x_2) = \ell_2$, then the challenger sends $\perp$ to $\mathcal{A}$. Otherwise, it computes polynomial $x_3 = x_1 \cdot x_2$. It executes an internal labeling query step for group $\mathbb{G}_T$ with input x_3 and sends $L_T(x_3)$ to $\mathcal{A}$.

Hash query: For each hash query, the challenger receives a string $s \in \{0,1\}^*$ from $\mathcal{A}$. If $s \notin H$, then the challenger sets $H(s) := X_s$, where X_s is an indeterminate, and executes an internal labeling query step for group $\mathbb{G}_1$ with input X_s. It sends $L_1(H(s))$ to $\mathcal{A}$.

Key computation query:

- $\mathcal{A}$ sends a list ids of B identities, $S = \{\mathsf{id}_1, \ldots, \mathsf{id}_B\} \subseteq \mathcal{I}$, along with a batch label $t \in \mathcal{T}$ to the challenger.
- If a key computation query has already been made with batch label t, the challenger halts the game.
- Otherwise, the challenger does the following:

- Let $f(X_\tau) = \sum_{i=0}^{B} f_i \cdot X_\tau^i$ be a univariate polynomial of degree B over $\mathbb{Z}_p$ with roots at $\mathsf{id}_1, \ldots, \mathsf{id}_B$ and leading coefficient 1. If $f(X_\tau) \notin L_1$, the challenger samples $z \leftarrow \{0,1\}^*$ and sets $L_1(f(X_\tau)) := z$.
- If $t \notin H$, it executes a hash query step with input t.
- Let $P := (f(X_\tau) + H(t)) \cdot X_{\mathsf{msk}}$ be a polynomial where the indeterminates are X_τ, $H(t)$ and X_{msk}. If $P \notin L_1$, the challenger samples $z \leftarrow \{0,1\}^*$ and sets $L_1(P) := z$.
- Finally, it sends the secret key $\mathsf{sk} := L_1(P)$ to $\mathcal{A}$.

Challenge round: Once during the game, $\mathcal{A}$ may decide that the current round is the *challenge round.* The challenge round proceeds as follows:

- $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^*$ and an identity-batch label pair (id^*, t^*) on which it wishes to be challenged.
- If key computation query (ids, t) has already been made with batch label $t = t^*$ and where $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.
- Otherwise, the challenger executes the following steps.
  - If $t^* \notin H$, execute a hash query step internally using input t^*.
  - Let $\mathbf{A}$ be a matrix and $\mathbf{b}$ be a vector defined as follows.

$$\mathbf{A} := \begin{pmatrix} 1 & \mathsf{id}^* - X_\tau & 0 \\ X_{\mathsf{msk}} & 0 & -1 \end{pmatrix}$$

$$\mathbf{b} := \begin{pmatrix} 0 \\ -H(t^*) \cdot X_{\mathsf{msk}} \end{pmatrix}$$

  - Let $\mathbf{X_r} := \begin{pmatrix} X_{r_1} \\ X_{r_2} \end{pmatrix}$ be a vector of two indeterminates. Compute a list $\mathbf{y}$ of four polynomials where

$$\mathbf{y} = (\mathbf{X_r}^T \cdot \mathbf{A}, \mathbf{X_r}^T \cdot \mathbf{b})$$

  - Parse $\mathbf{y}$ as (y_0, y_1, y_2, y_3). For each $i \in \{0,1,2,3\}$, set $L_2(y_i) \leftarrow \{0,1\}^*$.
  - Send the ciphertext $\mathbf{c} = (L_2(y_0), L_2(y_1), L_2(y_2), L_2(y_3 + L_2^{-1}(m_b)))$ to $\mathcal{A}$

Post-challenge queries: After the challenger round, $\mathcal{A}$ may again issue an arbitrary number of *key computation queries,* with the additional restriction that A cannot query (t^*, ids) with $\mathsf{id}^* \in \mathsf{ids}$:

- $\mathcal{A}$ sends a list ids of B identities along with a batch label t to the challenger.
- If a key computation query has already been made with batch label t **or if** $t = t^*$ **and** $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.

- Otherwise, the challenger executes the same steps as described in the Pre-challenge queries.

Output: At any point in time, $\mathcal{A}$ can decide to halt and output a bit $b' \in \{0,1\}$. The game then halts with the same output b'.

Proof. To prove the above lemma, we will introduce an intermediate hybrid experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Hyb},\mathsf{SM}}(1^\lambda, B)$ and show that for $b \in \{0,1\}$, the following holds:

$$\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE},\mathsf{SM}}(1^\lambda, B) \equiv \mathsf{Expt}_{\mathcal{A}}^{\mathsf{Hyb},\mathsf{SM}}(1^\lambda, B) \tag{1}$$

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Hyb},\mathsf{SM}}(1^\lambda, B)$ is same as $\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE},\mathsf{SM}}(1^\lambda, B)$ except that the fourth component of the ciphertext c in the challenge round is computed as follows:

- The challenger defines an indeterminate u and sets $L_2(u) \leftarrow S_2$ and updates $S_2 := S_2 \setminus L_2(u)$.
- The challenger sets the fourth component of ciphertext $\mathbf{c}$ to be $L_2(u)$.

To prove Eq. (1), we need to show that the polynomial y_3 involved in the ciphertext of challenge round of $\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE},\mathsf{SM}}(1^\lambda, B)$ is independent of the polynomials corresponding to all the other group element encodings which are in the view of adversary.

Without loss of generality, we will assume that the adversary makes n key computation queries with batch labels $t_1, \ldots, t_n$ and requests a challenge on $t_{i^*} = t^*$ for some $i^* \in [n]$ and identity id^*. We will use f^i to denote the degree B univariate polynomial having as roots the ids used in the i^{th} key computation query. Without loss of generality, we will assume that the hash queries are made on all the batch labels $t_1, \ldots, t_n$.

We will now list down the polynomials corresponding to the encodings held by the adversary.

$$L_1 = \left\{ 1, \{X_\tau^i\}_{i\in[B]}, \{H(t_i)\}_{i\in[n]}, \{(f^i(X_\tau) + H(t_i)) \cdot X_{\mathsf{msk}})\}_{i\in[n]} \right\}$$

$$L_2 = \left\{ 1, X_\tau, X_{\mathsf{msk}}, \underbrace{X_{r_1} + X_{r_2} \cdot X_{\mathsf{msk}}}_{\mathbf{r}^T \cdot a_1}, \underbrace{X_{r_1}(\mathsf{id}^* - X_\tau)}_{\mathbf{r}^T \cdot a_2}, \underbrace{-X_{r_2}}_{\mathbf{r}^T \cdot a_3} \right\}$$

The polynomial y_3 involved in $\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE},\mathsf{SM}}$ is $(-X_{r_2} \cdot H(t_{i^*}) \cdot X_{\mathsf{msk}})$ and we wish to show that it is outside the span of $L_1 \otimes L_2$, i.e., it is linearly independent of the list of polynomials obtained by multiplying polynomials in L_1 with polynomials in L_2. Let's assume, for the sake of contradiction, that this is not the case, i.e., y_3 happens to be a linear combination of polynomials in $L_1 \otimes L_2$. Then, by inspection, we have the following observations about the coefficients of the polynomials involved in the linear combination:

- Let $S_1 \subseteq L_1$ and $S_2 \subset L_2$ be the following sets:

$$S_1 = \left\{ 1, \{X_\tau^i\}_{i\in[B]}, \{H(t_i)\}_{i\in[n]}, \{(f^i(X_\tau) + H(t_i)) \cdot X_{\mathsf{msk}})\}_{i\in[n]} \right\}$$

$$S_2 = \left\{ 1, X_\tau, X_{\mathsf{msk}} \right\}$$

The coefficients of the polynomials in $S_1 \otimes S_2$ will be zero because the monomials in such polynomials do not occur in the target polynomial and are not present as monomials in other polynomials in$L_1 \otimes L_2$ (therefore they cannot be cancelled out).

- The coefficients of terms generated by the following completion will be zero for the same reason as above.

$$\left\{ 1, \{X_\tau^i\}_{i\in[B]}, \{H(t_i)\}_{i\in[n]} \right\} \otimes \left\{ \underbrace{-X_{r_2}}_{\mathbf{r}^T\cdot a_3} \right\}$$

- Similarly, the coefficients of the polynomials generated by the following completion will be zero as they contain monomials of the form $H(t_i)\cdot X_{r_2}\cdot X_{\mathsf{msk}}^2$ and $H(t_i)\cdot X_{\mathsf{msk}}\cdot X_{r_1}\cdot X_\tau$ which are neither present in the target polynomial nor in other polynomials in $L_1 \otimes L_2$.

$$\left\{ (f^i(X_\tau) + H(t_i))\cdot X_{\mathsf{msk}}) \}_{i\in[n]} \right\} \otimes \left\{ \underbrace{X_{r_1} + X_{r_2}\cdot X_{\mathsf{msk}}}_{\mathbf{r}^T\cdot a_1}, \underbrace{X_{r_1}(\mathsf{id}^* - X_\tau)}_{\mathbf{r}^T\cdot a_2} \right\}$$

- The coefficients of terms generated by the following completion will be zero.

$$\left\{ \{H(t_i)\}_{i\in[n]} \right\} \otimes \left\{ \underbrace{X_{r_1}(\mathsf{id}^* - X_\tau)}_{\mathbf{r}^\mathbf{T}\cdot a_2} \right\}$$

The reason is that it generates polynomials having monomials of the form $H(t_i)\cdot X_{r_1}\cdot X_\tau$. Since these monomials occur neither in the target polynomial, nor as monomials in the polynomials generated by other terms in $L_1 \otimes L_2$, their coefficients will be zero.

- The coefficients of terms generated by the following completion will be zero.

$$\left\{ \{H(t_i)\}_{i\in[n]} \right\} \otimes \left\{ \underbrace{X_{r_1} + X_{r_2}\cdot X_{\mathsf{msk}}}_{\mathbf{r}^\mathbf{T}\cdot a_1} \right\}$$

The polynomials generated by this completion contains monomials of the form $H(t_i)\cdot X_{r_1}$. The only other completion which generates this polynomial is $\left\{ \{H(t_i)\}_{i\in[n]} \right\} \otimes \left\{ \underbrace{X_{r_1}(\mathsf{id}^* - X_\tau)}_{\mathbf{r}^\mathbf{T}\cdot a_2} \right\}$. However, by previous observation, the coefficient of all terms of those completion are zero which, in turn, forces the coefficient of all terms in this completion to be also zero.

Finally, we are left with the following completion terms.

$$S_1 := \left\{ \{X_\tau^i\}_{i\in[B]} \right\} \otimes \left\{ \underbrace{X_{r_1} + X_{r_2} \cdot X_{\mathsf{msk}}}_{\mathbf{r}^T \cdot a_1}, \underbrace{X_{r_1}(\mathsf{id}^* - X_\tau)}_{\mathbf{r}^T \cdot a_2} \right\}$$

$$S_2 := \left\{ \{(f^i(X_\tau) + H(t_i)) \cdot X_{\mathsf{msk}})\}_{i\in[n]} \right\} \otimes \left\{ \underbrace{-X_{r_2}}_{\mathbf{r}^T \cdot a_3} \right\}$$

We note that each polynomial in S_2 has a monomial of the form $H(t_i) \cdot X_{\mathsf{msk}} \cdot X_{r_2}$. For all $i \in [n], i \neq i^*$, the coefficients of such polynomials would be zero as the monomial is neither present in the target polynomial nor in the other polynomials generated by the remaining completion. Hence, we are left with the following polynomials in the completion.

$$\left(\left\{ \{X_\tau^i\}_{i\in[B]} \right\} \otimes \left\{ \underbrace{X_{r_1} + X_{r_2} \cdot X_{\mathsf{msk}}}_{\mathbf{r}^T \cdot a_1}, \underbrace{X_{r_1}(\mathsf{id}^* - X_\tau)}_{\mathbf{r}^T \cdot a_2} \right\} \right) \bigcup \left\{ -(f^{i^*}(X_\tau) + H(t_{i^*})) \cdot X_{\mathsf{msk}}) \cdot X_{r_2} \right\}$$

$$= \left\{ \{X_\tau^i \cdot (X_{r_1} + X_{r_2} \cdot X_{\mathsf{msk}})\}_{i\in[B]}, \{X_\tau^i \cdot X_{r_1}(\mathsf{id}^* - X_\tau)\}_{i\in[B]}, -(f^{i^*}(X_\tau) + H(t_{i^*})) \cdot X_{\mathsf{msk}} \cdot X_{r_2} \right\}$$

Recall that the target polynomial y_3 is $(-X_{r_2} \cdot H(t_{i^*}) \cdot X_{\mathsf{msk}})$. Let $\{c_i, d_i\}_{i\in[0,B]}, e \in \mathbb{Z}_p$ be coefficients s.t.

$$\begin{aligned} -X_{r_2} \cdot H(t_{i^*}) \cdot X_{\mathsf{msk}} = &\sum_{i=0}^{B} c_i \cdot X_\tau^i \cdot (X_{r_1} + X_{r_2} \cdot X_{\mathsf{msk}}) \\ &+ \sum_{i=0}^{B} d_i \cdot X_\tau^i \cdot X_{r_1}(\mathsf{id}^* - X_\tau) \\ &- e(f^{i^*}(X_\tau) + H(t_{i^*})) \cdot X_{\mathsf{msk}} \cdot X_{r_2} \end{aligned}$$

From the above, it is clear that $e = 1$ to get the $(-X_{r_2} \cdot H(t_{i^*}) \cdot X_{\mathsf{msk}})$ monomial on the R.H.S. This implies that $\sum_{i=0}^{B} c_i \cdot X_\tau^i = f^{i^*}(X_\tau)$ so that the monomials involving $X_{r_2}, X_{\mathsf{msk}}$ vanish on the R.H.S. Hence, we have the following remaining constraint.

$$f^{i^*}(X_\tau) \cdot X_{r_1} = \sum_{i=0}^{B} d_i \cdot X_\tau^i \cdot X_{r_1} \cdot (X_\tau - \mathsf{id}^*)$$

The above constraint implies that id^* is a root of the polynomial $f^{i^*}(X_\tau)$ which is a contradiction as per the rules of the security game.

□

Lemma 5.4. *For all $B \in \mathbb{N}$ and all unbounded adversaries $\mathcal{A}$ making at most q queries (including queries to the group oracle, hash oracle, and key computation queries), for $b \in \{0, 1\}$, we have:*

$$\left| \Pr[\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE},\mathsf{GGM}}(1^\lambda, B) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},b}^{\mathsf{SBIBE},\mathsf{SM}}(1^\lambda, B) = 1] \right| \leq \binom{q + B + 5}{2} \frac{(B+2)}{p}$$

Proof. To prove this, we consider the following experiment[7]: At the end of $\mathsf{Expt}^{\mathsf{SBIBE,SM}}_{\mathcal{A},b}$, the challenger samples uniformly random values from $\mathbb{Z}_p$ for all the indeterminates involved in $\mathsf{Expt}^{\mathsf{SBIBE,SM}}_{\mathcal{A},b}$ and replaces all the polynomials with their evaluations. This is a perfect simulation of $\mathsf{Expt}^{\mathsf{SBIBE,GGM}}_{\mathcal{A},b}$ unless the following bad event happens: The sampled values result in a an identical evaluation of polynomials which are not identical. For any pair of polynomials (f_1, f_2), of total degree (d_1, d_2) respectively, we can bound the probability of this bad event happening to be at most $\frac{\max(d_1,d_2)}{p}$ using Schwartz-Zippel lemma [Sch80; Zip79]. There are $q+B+5$ polynomials in total across all three groups (q polynomials from queries and $B+5$ polynomials from the setup phase) with maximum total degree at most $B+2$. Therefore, union bounding across all possible pairs of polynomials gives us a maximum failure probability of $\binom{q+B+5}{2}\frac{(B+2)}{p}$. □

6 Extensions and Optimizations

6.1 Thresholdizing the scheme

In our Selective Batched IBE scheme as defined in Definition 4.1, the master secret-key msk is held by a central authority which runs $\mathsf{ComputeKey}$ procedure to derive the batch label specific secret keys sk and distribute them as necessary. In many applications, including the ones we have discussed in the introduction, it is favorable to distribute this trust among multiple authorities. Specifically, instead of a single authority holding the complete msk, we would like to have multiple, let's say n, authorities where each authority $i \in [n]$ holds a "partial" master secret key msk_i (e.g., in the form of secret shares of msk). In such a distributed setting, it is highly desirable to construct a scheme where multiple authorities can securely issue the batch label specific secret keys sk without leaking their "partial" master secret key msk_i. We observe that our construction in Fig. 1 readily admits such an efficient threshold version. This is due to the fact that the $\mathsf{ComputeKey}$ procedure in our construction simply computes a BLS-like signature which can be efficiently thresholdized as observed in prior works [Bol03]. For completeness, we define Thresholdizable Selective Batched Identity Based Encryption in Appendix A and show how our non-threshold construction can be modified to obtain a threshold version.

6.2 Outsourcing the digest computation

In our construction, the digest $d = \mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\})$ is computed as a KZG commitment to the polynomial $f(X) = \sum_{i=0}^{B} f_i \cdot X^i$ that has roots at $\mathsf{id}_1, \ldots, \mathsf{id}_B$ and leading coefficient 1. The digest is $d := \sum_{i=0}^{B} f_i \cdot [\tau^i]_1$.

We note that the work of computing the digest $d = \mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\})$ can be outsourced to a single server, such that the result can be efficiently verified by everyone. The main observation is that, since the polynomial $f(X)$ can be represented as $f(X) = \prod_{i=1}^{B}(X - \mathsf{id}_i)$ then for any value z in the field, the value $f(z)$ can be efficiently computed using only B multiplications, which is much faster than interpolating the polynomial and computing the digest.

Verification can thus be implemented using the Fiat-Shamir paradigm:

[7] We note that the master theorem proved in [BBG05] cannot be directly applied here as the adversary can make oracle queries other than standard group operations such as key computation queries. To account for this difference, we redo the analysis here which essentially follows the same style that is used in proving the master theorem.

- The server which computed the digest d computes a random field point $z = H(d)$, and computes $y = f(z)$.
- To provide an evaluation proof for the KZG commitment d, it computes the quotient polynomial $q(X) = \frac{f(X)-y}{X-z}$ and then computes and publishes a KZG evaluation proof $\pi = [q(\tau)]_1$.
- Given d and π, anyone can compute $z = H(d)$ and then efficiently compute $y = f(z)$. Then the proof can be verified by checking that $[1]_2 \circ (d - y) = ([\tau]_2 - [z]_2) \circ \pi$.

One can use the well-known Schwartz-Zippel lemma [Sch80; Zip79] to show that the above procedure satisifies soundness.

The work of computing the group multiplications in $d := \sum_{i=0}^{B} f_i \cdot [\tau^i]_1$ can also be distributed among m servers. The result can be efficiently verified by everyone, given that they have access to the polynomial $f(X)$. (Computing $f(X)$ itself is non-trivial, as it is essentially computing an interpolation.)

Assume that this work is distributed between m servers such that server $k = 1, \ldots, m$ computes $d_k := \sum_{i=(k-1)\cdot B/m}^{k\cdot B/m-1} f_i \cdot [\tau^i]_1$. (To simplify the notation we assume that B/m is an integer.) Then, given these results from the servers it is easy to compute $d = \sum_{k=1}^{m} d_k$. The work of each server is roughly $1/m$ the work of computing d. The main remaining issue is how to efficiently verify that each d_k is computed correctly (or, in the case of outsourcing to a single server, verifying that it computed d correctly). We note that server k actually computes a KZG commitment d_k for the polynomial $f_k = \sum_{i=(k-1)B/m}^{kB/m-1} f_i \cdot X^i$. This polynomial is of degree $\frac{kB}{m} - 1 \leq B$. Therefore d_k can be verified using the same procedure outlined above for verifying d.[8]

6.3 Batching the Decryption Procedure

For some applications, such as mempool privacy, the decryption of all ciphertexts in a batch will be done by a single party. A naive application of Decrypt will result in a total decryption time of $O(B^2 \log B)$. We discuss below how to improve this time. Note that in this section, we refer to "time" as the total number of group and field operations required to perform a task.

Note that the computation that dominates the running time for decryption is the computation of KZG commitment opening proofs. The time for computing all the B proofs naively is $O(B^2 \log B)$, whereas the time for the remaining work after the opening proofs are finished is simply $O(B)$. The work of [FK23] shows how to compute a set of opening proofs for a KZG commitment much more efficiently than the naive approach. Specifically, assume as in our decryption procedure we have a KZG commitment to a polynomial of degree B, and we need opening proofs for each of its B roots. [FK23] show that if the polynomial has been constructed so the roots are all part of some set of roots of unity Ω, then it is possible to compute all openings in

[8] It is possible to do an efficient *optimistic* verification of the work of the servers, in the sense that if all servers are honest then the result can be efficiently verified without interpolating $f(X)$. Otherwise, after interpolating $f(X)$ it is possible to identify which server k provided an incorrect d_k value. The proof process is as follows. First, a value z is computed as $H(d_1, \ldots, d_m)$. Then each server provides $y_k = f_k(z)$ and a proof that the polynomial committed to by d_k has the output y_k at the point z. The verifier computes $f(z)$ and verifies that it is equal to $\sum_{k=1}^{m} y_k$. If $d = \sum_{k=1}^{m} d_k$ is not a commitment to $f(X)$, the by the Schwartz-Zippel theorem this check fails with all but negligible probability. (Note that this is not a check that each d_k is computed correctly, but rather that $d = \sum_{k=1}^{m} d_k$ is correct, which is the property that we need.

time $O(|\Omega| \log|\Omega|)$. Specifically, if we want to support a maximum batch size B, we can set Ω such that $|\Omega| = B$, and then as long as we choose the IDs from Ω, we can batch-decrypt in time $O(B \log B + B) = O(B \log B)$. In addition, [FK23] show that even if the roots of the polynomial are chosen arbitrarily, it is still possible to compute the B openings in time $O(|\Omega| \log^2|\Omega|)$. So if the particular application requires more flexibility in choosing IDs, it is still possible to batch-decrypt in time $O(B \log^2 B + B) = O(B \log^2 B)$.

The batch decryption procedure BatchDecrypt**.**

BatchDecrypt$(\{c_1, \ldots, c_B\}, \mathsf{sk}, d, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}, \mathsf{id}, t)$**:**

- **(KZG opening proof computation)** Use [FK23] to derive openings $\pi_1, \ldots, \pi_B$ for the roots $\{\mathsf{id}_1, \ldots, \mathsf{id}_B\}$ of the polynomial $\pi_i\,(X - \mathsf{id}_i)$, with respect to digest d.
- **(Decryption)** For each $i \in [B]$:
  1. Set
  $$\mathbf{w}_i = \begin{pmatrix} d \\ \pi_i \\ \mathsf{sk} \end{pmatrix}.$$
  2. Parse c_i as $(c_{i,1}, c_{i,2})$, and set $m_i = c_{i,2} - c_{i,1} \circ \mathbf{w}$.
- Output $\{m_1, \ldots, m_B\}$.

6.4 Non-Malleability for Mempool Privacy

As mentioned in [Cho+24], the application of mempool privacy has a specific requirement that ciphertexts need to be *authenticated,* and must satisfy a form of non-malleability. Specifically, a ciphertext encrypts a signed transaction coming from a specific sender address; this address is associated with the public key of the signature on the transaction. Only the sender, who possesses the signing key, is allowed to submit the ciphertext for decryption, and if an encrypted transaction has not yet made it to a block, the transaction details must be hidden from any adversary who does not possess the secret key for the transaction sender address, even if this adversary can submit arbitrary decryption requests on any other ciphertext from any other sender address.

The authors of [Cho+24] state that "adding non-malleability to ciphertexts corresponds closely to securing [the] encryption scheme against chosen ciphertexts." We point out that achieving the form of non-malleability required here is actually *weaker* than CCA security. This is because CCA security does not associate ciphertexts with signature public keys, and requires indistinguishability even if an adversary can submit arbitrary ciphertexts for decryption, as long as they are not equal to the challenge ciphertext. In our setting, ciphertexts are inherently associated with a signature public key, since they encrypt a signed message under that public key, and our decryption oracle is more restrictive: it only decrypts messages that are signed, and we assume that the adversary does not have the signing key corresponding to the challenge ciphertext.

We can use this fact to achieve non-malleability significantly more cheaply than [Cho+24], which uses NIZK proofs to add full CCA2-security to their encryption. In contrast, we can simply use a signature scheme. We will use the decryption functionality below. At a high level, the

decryption functionality works as follows, assuming it has access to whatever signature scheme is being used to sign transactions on the chain. First, it associates every ID with a signature public key. We do this by setting the ID to be a hash of the public key, i.e., $\mathsf{id} = H(\mathsf{pk})$. But the functionality could also maintain a table of ID-public key mappings. Second, whenever the decryption functionality takes as input a ciphertext ct to be decrypted during block t with associated id id, it requires a signature on the block number t that is valid with respect to the public key $\mathsf{pk} = H^{-1}(\mathsf{id})$. If this signature fails to verify, then the functionality refuses to add this ID to the set of approved IDs to be decrypted during block t.

We note that for simplicity, the functionality as written assumes that each party submits at most one encrypted transaction per block from their address. It is easy to extend to the more general case, as follows: for each transaction to be submitted, the sender chooses a nonce x, and sets the ID of the ciphertext to be $\mathsf{id} = H(\mathsf{pk}, x)$. Then, when accepting transactions, the functionality requires a signature on message (t, x) instead of just t.

We now describe the decryption functionality formally.

The decryption functionality.

Setup: We assume the decryption functionality is initialized with a SBIBE keypair $(\mathsf{pk}^{\mathsf{SBIBE}}, \mathsf{msk})$ and block size B. We also assume the functionality has access to a random oracle H which maps arbitrary strings to the ID space of SBIBE. This functionality is stateful, and maintains lists A_t of IDs that have been authorized to be decrypted during block t.

Decryption authorization: The functionality takes in messages of the form $\langle \mathsf{id}, t, \mathsf{pk}, \mathsf{ct}, \sigma \rangle$, which indicates that the party holding the blockchain account corresponding to public key pk is authorizing decryption of ct during block t. The functionality does the following:

1. Check that $\mathsf{id} = H(\mathsf{pk})$. If not, halt and respond with $\perp$.
2. Check that σ is a valid signature over the message t with public key pk. If not, halt and respond with $\perp$.
3. Add id to the list A_t to record the id id as authorized to decrypt during block t.

Block transactions decryption: The functionality also handles block decryption queries of the form $\langle t, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\} \rangle$. It does the following:

1. If there is an i such that id_i is not in the list A_t, halt and respond with $\perp$.
2. Compute the digest $d \leftarrow \mathsf{Digest}(\mathsf{pk}^{\mathsf{SBIBE}}, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\})$.
3. Compute the decryption key $\mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, d, t)$ which allows for decryption of ciphertexts encrypted to any id in $\{\mathsf{id}_1, \ldots, \mathsf{id}_B\}$ for block t.
4. Respond with sk.

We define the security game below, where an adversary attempts to distinguish against a ciphertext that has not yet made it into a block.

The security game $\mathsf{Expt}^{\mathsf{mempool}}_{\mathcal{A},b}(1^\lambda, B)$**.**

Setup:

1. Generate SBIBE keypair $\mathsf{msk}^{\mathsf{SBIBE}}, \mathsf{pk}^{\mathsf{SBIBE}}$.
2. Generate transaction signing keypair sk, pk which will be used to sign the challenge plaintext.
3. Send $\mathsf{pk}^{\mathsf{SBIBE}}$ and pk to $\mathcal{A}$.

Handling queries: At any time, both before and after the challenge round, $\mathcal{A}$ can submit decryption authorization and block transaction decryption queries, and the challenger responds to them exactly as defined in the functionality above. **In addition**, $\mathcal{A}$ can ask for a signature on any message t with respect to the signing key sk.

Challenge: At any time $\mathcal{A}$ can send (t^*, m_0, m_1) and ask for the challenge ciphertext, with the restriction that t^* has not already passed, and that $\mathcal{A}$ has not queried for a signature on t^*. The challenger will then sign m_b with sk to get σ, generate $\mathsf{id} \leftarrow H(\mathsf{pk})$, and then encrypt using $\mathsf{Encrypt}(\mathsf{pk}^{\mathsf{SBIBE}}, (m_b, \sigma), \mathsf{id}, t^*)$ and send the result to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit b'.

Claim. *Let $\mathcal{A}$ be a PPT adversary. Then assuming* SBIBE *is secure under Definition 4.4, and the signature scheme satisfies standard existential unforgeability, then*

$$\left|\Pr[\mathsf{Expt}^{\mathsf{mempool}}_{\mathcal{A},0}(1^\lambda, B) = 1] - \Pr[\mathsf{Expt}^{\mathsf{mempool}}_{\mathcal{A},1}(1^\lambda, B) = 1]\right| < \epsilon_A(\lambda).$$

Proof sketch. Assume there is an $\mathcal{A}$ which can distinguish between the two experiments. This means it distinguishes a ciphertext encrypted under SBIBE which has been encrypted under $\mathsf{id} = H(\mathsf{pk})$, and round t^* where pk is a signature scheme public key whose corresponding signing key is not known to $\mathcal{A}$, and where $\mathcal{A}$ has not received a signature over t^*.

There are two cases, conditioned on whether id was included in a decryption key $\mathsf{sk}^{\mathsf{SBIBE}}$ which was given to $\mathcal{A}$ from the decryption functionality.

First, assume that id was indeed included in some decryption key. Recall from the definition of the decryption functionality that this would only happen if $\mathcal{A}$ was able to provide a signature on t^*. This contradicts existential unforgeability of the signature scheme.

Assume now that id was never included in any decryption key $\mathsf{sk}^{\mathsf{SBIBE}}$ given to $\mathcal{A}$. Then by distinguishing, $\mathcal{A}$ directly contradicts security of the SBIBE scheme as defined in Definition 4.4. □

7 Concrete Performance

In this section, we discuss the concrete performance of our scheme. We have implemented our Selective Batched IBE scheme in rust, using the `arkworks` framework [con22] and the BLS12-381 curve. The benchmarks were run using a Google Cloud VM of type `t2d-standard-4`, with a four-core AMD EPYC Milan CPU and 16GB of RAM. Table 2 shows the time taken to compute a

digest, to compute a decryption key, to encrypt, and to decrypt, with respect to several different batch sizes. We have omitted benchmarking the setup time, since our setup is simply a KZG setup plus a BLS key-generation, both of which are standard and have been well-studied.

Batch Size	Digest Computation	Key Computation	Encryption	Decryption
100	11.5ms	720µs	6.4ms	9.1ms
1,000	104.4ms	643µs	6.5ms	88.7ms
10,000	877ms	681µs	6.4ms	778.5ms
100,000	8.6s	759µs	6.4ms	8.6s

Table 2: Running times of each procedure in our scheme for varying batch sizes.

Mempool Privacy: Comparison with [Cho+24] and related works In addition to implementing the vanilla version of our scheme, we also implemented the extensions in Sections 6.1, 6.3 and 6.4, in order to compare the performance of our scheme with that of [Cho+24]. That is, we implemented a version of our scheme with threshold decryption key computation, with batched decryption using [FK23], and with signature verification over the submitted ciphertexts. We used the `ed25519-dalek` library for signatures.

Batch Size	[Cho+24] w/o Setup	[Cho+24] w/ Setup	Ours
8	83.218ms	18.08s	30.96ms
32	337.5ms	18.34s	114.7ms
128	1.422s	19.42s	462.1ms
512	6.02s	24.02s	1.92s

Table 3: Total time to decrypt by batch size: comparison with [Cho+24].

In order to get an accurate comparison, we used the code from [Cho+24][9] to re-run their benchmarks using the same Google Cloud VM of type `t2d-standard-4` which we used for benchmarking our scheme. We instantiated our scheme with the same threshold parameters ($n = 16, t = 4$) as in their benchmark, and with IDs chosen as roots of unity to enable the fast version of [FK23] during decryption. Section 7 below gives, for both our scheme and theirs, the total time for a single batch decryption. Recall that [Cho+24] require an expensive, per-batch-decryption setup phase. The MPC protocol for computing this setup phase was not implemented by them, as far as we know, but [Cho+24] estimates in their paper that it would take around 18 seconds. We have included the time for their scheme both with and without this setup cost.

For completeness, in Table 4, we also provide a comparison of asymptotic communication and computation cost for performing selective batched threshold decryption. We also mention whether a per batch setup phase is required. Note that all the mentioned approaches require a global one-time setup phase, as is typical in threshold cryptosystems, so we don't show this explicitly in the table.

[9] URL is `https://github.com/guruvamsi-policharla/batched-threshold-encryption`.

Scheme	Comm.	Computation		Per batch setup phase required ?
		Total	Private	
[ElG86; CG99]	$O(nB)$	$O(B)$	$O(B)$	No
Ferveo [BO22]	$O(nB)$	$O(B)$	$O(B)$	No
Choudhuri et. al. [Cho+24]	$O(n)$	$O(B \log B)$	$O(B \log B)$	Yes
This work	$O(n)$	$O(B \log B)$	$O(1)$	No

Table 4: Comparison of the costs required for performing *selective* batched threshold decryption for a batch size B and n servers. The communication and computation costs are per server and are represented in terms of the number of group elements and group operations respectively. It also shows whether the servers need to perform a setup phase for every batch.

8 Acknowledgements

We would like to thank Alin Tomescu and Andrei Tonkikh for many useful discussions related to this project.

References

[Bau+23] Balthazar Bauer, Pooya Farshim, Patrick Harasser, and Markulf Kohlweiss. "The uber-knowledge assumption: A bridge to the AGM". In: *Cryptology ePrint Archive* (2023).

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. "Hierarchical identity based encryption with constant size ciphertext". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005, pp. 440–456.

[BF01] Dan Boneh and Matt Franklin. "Identity-based encryption from the Weil pairing". In: *Annual international cryptology conference*. Springer. 2001, pp. 213–229.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short signatures from the Weil pairing". In: *International conference on the theory and application of cryptology and information security*. Springer. 2001, pp. 514–532.

[BO22] Joseph Bebel and Dev Ojha. "Ferveo: Threshold decryption for mempool privacy in bft networks". In: *Cryptology ePrint Archive* (2022).

[Bol03] Alexandra Boldyreva. *Efficient threshold signature, multisignature and blind signature schemes based on the Gap-Diffie-Hellman-group signature scheme, PKC 2003, LNCS 2139*. 2003.

[Cam+21] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. *Encryption to the Future: A Paradigm for Sending Secret Messages to Future (Anonymous) Committees*. Cryptology ePrint Archive, Paper 2021/1423. 2021. URL: https://eprint.iacr.org/2021/1423.

[Cer+23] Andrea Cerulli, Aisling Connolly, Gregory Neven, Franz-Stefan Preiss, and Victor Shoup. "Vetkeys: How a blockchain can keep many secrets". In: *Cryptology ePrint Archive* (2023).

[CG99] Ran Canetti and Shafi Goldwasser. "An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1999, pp. 90–106.

[Chi+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Advances in Cryptology - EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 738–768.

[Cho+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. "Fairness in an unfair world: Fair multiparty computation from public bulletin boards". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 719–728.

[Cho+24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. "Mempool Privacy via Batched Threshold Encryption: Attacks and Defenses". In: *Cryptology ePrint Archive* (2024).

[Cle86] Richard Cleve. "Limits on the security of coin flips when half the processors are faulty". In: *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. 1986, pp. 364–369.

[con22] Arkworks contributors. *`arkworks` zkSNARK ecosystem*. 2022. URL: `https://arkworks.rs`.

[Dai+20] 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 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 910–927. DOI: `10.1109/SP40000.2020.00040`. URL: `https://doi.org/10.1109/SP40000.2020.00040`.

[Dal+20] Anders Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. "Securing DNSSEC keys via threshold ECDSA from generic MPC". In: *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part II 25*. Springer. 2020, pp. 654–673.

[Dam+12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. "Multiparty computation from somewhat homomorphic encryption". In: *Annual Cryptology Conference*. Springer. 2012, pp. 643–662.

[Döt+23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. "McFly: verifiable encryption to the future made practical". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2023, pp. 252–269.

[ElG86] Taher ElGamal. "On computing logarithms over finite fields". In: *Advances in Cryptology—CRYPTO'85 Proceedings 5*. Springer. 1986, pp. 396–402.

[FK23] Dankrad Feist and Dmitry Khovratovich. "Fast amortized KZG proofs". In: *IACR Cryptol. ePrint Arch.* (2023), p. 33. URL: `https://eprint.iacr.org/2023/033`.

[Gar+13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. "Witness encryption and its applications". In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013, pp. 467–476.

[Gar+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. "Candidate indistinguishability obfuscation and functional encryption for all circuits". In: *SIAM Journal on Computing* 45.3 (2016), pp. 882–929.

[Gar+24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. "Threshold Encryption with Silent Setup". In: *Annual International Cryptology Conference*. Springer. 2024, pp. 352–386.

[GMR23] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. "tlock: Practical timelock encryption from threshold bls". In: *Cryptology ePrint Archive* (2023).

[Gol09] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2009.

[Kav+23] Alireza Kavousi, Duc V. Le, Philipp Jovanovic, and George Danezis. *BlindPerm: Efficient MEV Mitigation with an Encrypted Mempool and Permutation*. Cryptology ePrint Archive, Paper 2023/1061. 2023. URL: https://eprint.iacr.org/2023/1061.

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. "Constant-size commitments to polynomials and their applications". In: *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer. 2010, pp. 177–194.

[LPS23] Helger Lipmaa, Roberto Parisella, and Janno Siim. "Algebraic group model with oblivious sampling". In: *Theory of Cryptography Conference*. Springer. 2023, pp. 363–392.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. *Time-lock puzzles and timed-release Crypto*. Technical Report MIT-LCS-TR-684. Massachusetts Institute of Technology, 1996. URL: https://people.csail.mit.edu/rivest/pubs/RSW96.pdf.

[SA19] Nigel P Smart and Younes Talibi Alaoui. "Distributing any Elliptic Curve Based Protocol: With an Application to MixNets." In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 768.

[Sch80] Jacob T Schwartz. "Fast probabilistic algorithms for verification of polynomial identities". In: *Journal of the ACM (JACM)* 27.4 (1980), pp. 701–717.

[Sho97] Victor Shoup. "Lower bounds for discrete logarithms and related problems". In: *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*. Springer. 1997, pp. 256–266.

[Tsa22] Rotem Tsabary. "Candidate witness encryption from lattice techniques". In: *Annual International Cryptology Conference*. Springer. 2022, pp. 535–559.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. "Witness encryption and null-IO from evasive LWE". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2022, pp. 195–221.

[Zha22] Mark Zhandry. "To label, or not to label (in generic groups)". In: *Annual International Cryptology Conference*. Springer. 2022, pp. 66–96.

[Zip79] Richard Zippel. "Probabilistic algorithms for sparse polynomials". In: *International symposium on symbolic and algebraic manipulation*. Springer. 1979, pp. 216–226.

A Thresholdizable Selective Batched Identity Based Encryption

In this section, we describe a threshold version of the Selective Batched IBE scheme which we defined in Definition 4.1. At a high-level, this threshold version is aimed to capture settings where one would like to distribute the trust of key issuance across multiple authorities. Specifically, instead of a single authority holding the complete msk, we would like to have multiple, let's say n, authorities where each authority $i \in [n]$ holds a "partial" master secret key msk_i (e.g., in the form of secret shares of msk). Accordingly, the $\mathsf{ComputeKey}$ procedure (as defined in Selective Batched IBE) will be split into two parts: 1) $\mathsf{ComputeKeyShare}$ algorithm which will be used by each authority to produce a *partial* secret key w.r.t a specific batch using its private partial master secret key msk_i, 2) $\mathsf{ComputeKeyAggregate}$ algorithm which can be used to combine the partial secret key w.r.t a specific batch into a full secret key.

In the following sections, we will formally define the syntax and semantics of Thresholdizable Selective Batched IBE . We will then present a construction of Thresholdizable Selective Batched IBE (which is a straightforward adaptation of our Selective Batched IBE construction) and analyze it. For the ease of readability, we will highlight all the differences between Thresholdizable Selective Batched IBE and Selective Batched IBE in blue.

A.1 Syntax

Definition A.1 (Thresholdizable Selective Batched IBE Syntax)**.** A Thresholdizable Selective Batched IBE scheme TSBIBE is specified by seven algorithms: Setup, KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$, Digest, $\mathsf{ComputeKeyShare}$, $\mathsf{ComputeKeyAggregate}$.

- $\mathsf{Setup}(1^\lambda, 1^B, 1^n) \to \mathsf{params}$: A randomized algorithm that takes as input a security parameter $\lambda \in \mathbb{N}$, a batch size $B = B(\lambda)$ and number of authorities $n = n(\lambda)$. It outputs params (system parameters) which includes a description of the message space $\mathcal{M}$, identity space $\mathcal{I}$, batch label space $\mathcal{T}$ and ciphertext space $\mathcal{C}$.
- $\mathsf{KeyGen}(\mathsf{params}) \to (\{\mathsf{msk}_i\}_{i\in[n]}, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{pk})$: a randomized algorithm that takes as input params and outputs n many msk_i (partial master secret key), n many pk_i (partial public key) and a single pk (global public key).
- $\mathsf{Encrypt}(\mathsf{pk}, m, \mathsf{id}, t) \to c$: a randomized algorithm that takes as input a message $m \in \mathcal{M}$, an identity $\mathsf{id} \in \mathcal{I}$, a batch label $t \in \mathcal{T}$, global public key pk and outputs a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}) \to d$: a deterministic algorithm that takes as input the global public key pk and a list of identities $\mathsf{id}_1, \ldots, \mathsf{id}_B$ where each $\mathsf{id}_i \in \mathcal{I}$. It outputs a digest d.
- $\mathsf{ComputeKeyShare}(\mathsf{msk}_i, d, t) \to \mathsf{sk}_i$: a deterministic algorithm that takes as input the partial master secret key msk_i, digest d, batch label t and outputs a partial digest-batch label-specific secret key sk_i.

- $\mathsf{ComputeKeyAggregate}(\{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in[n]}, d, t) \to \mathsf{sk}$: a deterministic algorithm that takes as input all the partial public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ and all the partial digest-batch label-specific secret key $\{\mathsf{sk}_i\}_{i\in[n]}$ and outputs a digest-batch label-specific secret key sk.
- $\mathsf{Decrypt}(c, \mathsf{sk}, d, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}, \mathsf{id}, t) \to m$: a deterministic algorithm that takes as input a ciphertext c, secret key sk, digest d, a list of identities $\mathsf{id}_1, \ldots, \mathsf{id}_B$ and an identity-batch label pair (id, t). It outputs a message $m \in \mathcal{M}$.

A.2 Correctness, Non-triviality and Security

The above algorithms should satisfy the following requirements.

For correctness, we generalize the correctness requirement of the (non-threshold) Selective Batched IBE to allow the adversary to (statically) corrupt at most half the number of authorities, get their partial master secret-keys, and observe the partial secret-keys issued by uncorrupted authorities for arbitrary inputs of the adversary's choice.

Definition A.2 (Thresholdizable Selective Batched IBE Correctness)**.** For all $\lambda \in \mathbb{N}, B \in \mathbb{N}, n \in \mathbb{N}, m \in \mathcal{M}, t \in \mathcal{T}, \mathsf{id} \in \mathcal{I}, S \subseteq \mathcal{I}$ s.t. $|S| = B$ and $\mathsf{id} \in S$, $\mathsf{Cor} \subset [n]$ s.t. $|\mathsf{Cor}| \leq \left\lfloor \frac{n-1}{2} \right\rfloor$ and for any unbounded adversary $\mathcal{A}$, the following should hold:

$$\Pr\left[\mathsf{Decrypt}(c, \mathsf{sk}, d, S, \mathsf{id}, t) = m \;\middle|\; \begin{array}{c} \mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^B, 1^n) \\ (\{\mathsf{msk}_i\}_{i\in[n]}, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{params}) \\ c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m, \mathsf{id}, t) \\ d \leftarrow \mathsf{Digest}(\mathsf{pk}, S) \\ \forall i \in [n] \setminus \mathsf{Cor}, \mathsf{sk}_i \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_i, d, t) \\ \forall i \in \mathsf{Cor}, \mathsf{sk}_i \leftarrow \mathcal{A}^{\{\mathsf{ComputeKeyShare}(\mathsf{msk}_k, \cdot, \cdot)\}_{k\in[n]\setminus\mathsf{Cor}}}(\{\mathsf{msk}_j\}_{j\in\mathsf{Cor}}, \\ \{\mathsf{pk}_j\}_{j\in[n]\setminus\mathsf{Cor}}, \mathsf{pk}, m, \mathsf{id}, t, c, S) \\ \mathsf{sk} \leftarrow \mathsf{ComputeKeyAggregate}(\{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in[n]}, d, t) \end{array}\right] = 1$$

Definition A.3 (Thresholdizable Selective Batched IBE Non-triviality/Efficiency)**.** We require that the running time of $\mathsf{ComputeKeyShare}$ and $\mathsf{ComputeKeyAggregate}$ be independent of the batch size B (which implies that the digest d and sk are also independent of B). We also require that the running time of $\mathsf{ComputeKeyShare}$ and the size of sk be independent of the number of authorities n.

For security, we generalize the security requirement of the (non-threshold) Selective Batched IBE to allow the adversary to (statically) corrupt at most half the number of authorities, get their partial master secret-keys, and observe the partial secret-keys issued by uncorrupted authorities for arbitrary inputs of the adversary's choice while constrained to the same rules as defined earlier in the non-threshold version.

Definition A.4 (Thresholdizable Selective Batched IBE Security)**.** We define a security game $\mathsf{Expt}^{\mathsf{TSBIBE}}_{\mathcal{A},b}(1^\lambda, B, n)$ with respect to adversary $\mathcal{A}$ in the box below.

We say that a Thresholdizable Selective Batched IBE scheme is secure if for all $n \in \mathbb{N}$, $B \in \mathbb{N}$, for all PPT adversaries $\mathcal{A}$, for all $\mathsf{Cor} \subset [n]$ s.t. $|\mathsf{Cor}| \leq \left\lfloor \frac{n-1}{2} \right\rfloor$, there exists some negligible function $\epsilon_{\mathcal{A}}$ such that the following holds:

$$\left|\Pr[\mathsf{Expt}^{\mathsf{TSBIBE}}_{\mathcal{A},0}(1^\lambda, B, n, \mathsf{Cor}) = 1] - \Pr[\mathsf{Expt}^{\mathsf{TSBIBE}}_{\mathcal{A},1}(1^\lambda, B, n, \mathsf{Cor}) = 1]\right| < \epsilon_{\mathcal{A}}(\lambda).$$

The security game $\mathsf{Expt}^{\mathsf{TSBIBE}}_{\mathcal{A},b}(1^\lambda, B, n, \mathsf{Cor})$.

Setup: The challenger takes as input the security parameter λ and the batch size B. It runs $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^B, 1^n)$, and then runs
$(\{\mathsf{msk}_i\}_{i\in[n]}, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$.
Finally, it sends $(\mathsf{params}, \mathsf{pk}, \{\mathsf{msk}_i\}_{i\in\mathsf{Cor}}, \{\mathsf{pk}_i\}_{i\in[n]})$ to $\mathcal{A}$.

The rest of the game proceeds in rounds, as follows.
Pre-challenge queries: $\mathcal{A}$ may issue an arbitrary number of *key computation queries:*

- $\mathcal{A}$ sends a list ids of B identities along with a batch label t to the challenger.
- If a key computation query has already been made with batch label t, the challenger halts the game.
- Otherwise, the challenger does the following:
  - Compute a digest $d \leftarrow \mathsf{Digest}(\mathsf{pk}, \mathsf{ids})$ of the ids in ids using public key pk.
  - For all $i \in [n] \setminus \mathsf{Cor}$, compute a partial secret key $\mathsf{sk}_i \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_i, d, t)$, using the digest d computed from the previous step.
  - Send $\{\mathsf{sk}_i\}_{i\in[n]\setminus\mathsf{Cor}}$ to $\mathcal{A}$.

Challenge round: Once during the game, $\mathcal{A}$ may decide that the current round is the *challenge round.* The challenge round proceeds as follows:

- $\mathcal{A}$ sends two messages $m_0, m_1 \in \mathcal{M}$ and an identity-batch label pair (id^*, t^*) on which it wishes to be challenged.
- If key computation query (ids, t) has already been made with batch label $t = t^*$ and where $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.
- Otherwise, the challenger computes $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b, \mathsf{id}^*, t^*)$ and sends c to $\mathcal{A}$.

Post-challenge queries: After the challenger round, $\mathcal{A}$ may again issue an arbitrary number of *key computation queries,* with the additional restriction that A cannot query (t^*, ids) with $\mathsf{id}^* \in \mathsf{ids}$:

- $\mathcal{A}$ sends a list ids of B identities along with a batch label t to the challenger.
- If a key computation query has already been made with batch label t **or if** $t = t^*$ **and** $\mathsf{id}^* \in \mathsf{ids}$, the challenger halts the game.

- Otherwise, the challenger does the following:
  - Compute a digest $d \leftarrow \mathsf{Digest}(\mathsf{pk}, \mathsf{ids})$ of the ids in ids using public key pk.
  - For all $i \in [n] \setminus \mathsf{Cor}$, compute a partial secret key $\mathsf{sk}_i \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_i, d, t)$, using the digest d computed from the previous step.
  - Send $\{\mathsf{sk}_i\}_{i \in [n] \setminus \mathsf{Cor}}$ to $\mathcal{A}$.

Output: At any point in time, $\mathcal{A}$ can decide to halt and output a bit $b' \in \{0, 1\}$. The game then halts with the same output b'.

A.3 Construction

Theorem A.5. *Assuming Type-3 pairing group $\mathcal{BG}$, there exists a construction (Fig. 2) for Thresholdizable Selective Batched IBE which is secure in the Generic group model.*

A.4 Correctness

The correctness follows directly from the correctness of our non-threshold version of the scheme along with the robustness property of threshold BLS signatures [Bol03].

A.5 Security

We will prove the security of our construction in the GGM model equipped with oblivious sampling. In this model, the challenger will implement the group oracle and the hash oracle (along with an oracle for key computation queries as defined in the security game).

Theorem A.6. *For all $p = O(2^\lambda)$, $B = \mathsf{poly}(\lambda), n = \mathsf{poly}(\lambda)$, for all $\mathsf{Cor} \subset [n]$ s.t. $|\mathsf{Cor}| \leq \lfloor \frac{n-1}{2} \rfloor$, and all unbounded adversaries $\mathcal{A}$ making at most $q = \mathsf{poly}(\lambda)$ queries (including queries to the group oracle, hash oracle, and key computation queries), we have*

$$\left| \Pr[\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{TSBIBE},\mathsf{GGM}}(1^\lambda, B, n, \mathsf{Cor}) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},1}^{\mathsf{TSBIBE},\mathsf{GGM}}(1^\lambda, B, n, \mathsf{Cor}) = 1] \right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Expt}^{\mathsf{TSBIBE},\mathsf{GGM}}$ refers to the same experiment $\mathsf{Expt}^{\mathsf{TSBIBE}}$ as defined in Definition A.4 except that we specialize it for our specific Construction (Fig. 2) and model it in the GGM, i.e., all the group and hash operations performed by the adversary are simulated by the challenger as defined in the GGM model (Section 3.2). This is done in a manner similar to $\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}$ in Thm. 5.2.

Proof. To prove the above theorem, we will show that the security of the threshold version of our construction w.r.t $\mathsf{Expt}^{\mathsf{TSBIBE},\mathsf{GGM}}$ can be reduced to the security of the non-threshold version of our construction w.r.t. $\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}$.

Assume, for the sake of contradiction, that Thm. A.6 is false. Then, there exists $p = O(2^\lambda), B = \mathsf{poly}(\lambda), n = \mathsf{poly}(\lambda)$, $\mathsf{Cor} \subset [n]$ s.t. $|\mathsf{Cor}| \leq \lfloor \frac{n-1}{2} \rfloor$, there exists an adversary $\mathcal{A}$ making at most q queries s.t. there exists a polynomial $\mathsf{poly}_\mathcal{A}$ where

$$\left| \Pr[\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{TSBIBE},\mathsf{GGM}}(1^\lambda, B, n, \mathsf{Cor}) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},1}^{\mathsf{TSBIBE},\mathsf{GGM}}(1^\lambda, B, n, \mathsf{Cor}) = 1] \right| > \frac{1}{\mathsf{poly}_\mathcal{A}(\lambda)}$$

- $\mathsf{Setup}(1^\lambda, 1^B, 1^n)$: Output three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order p, where p is a λ-bit prime, equipped with generators g_1, g_2, g_T, respectively, and an efficiently computable pairing operation $\circ : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$. Set the message space $\mathcal{M} := \mathbb{G}_T$, identity space $\mathcal{I} := \{0, \ldots, p-1\}$, and tag space $\mathcal{T} := \{0,1\}^\lambda$. Also output a randomly sampled hash function $H : \mathcal{T} \to \mathbb{G}_1$.

- $\mathsf{KeyGen}(\mathsf{params})$: Sample $\mathsf{msk} \leftarrow \mathbb{Z}_p$ and $\tau \leftarrow \mathbb{Z}_p$. Set $\{\mathsf{msk}_i\}_{i\in[n]} \leftarrow \mathsf{ShamirShare}(\mathsf{msk}, \lfloor \frac{n-1}{2} \rfloor, n)$, i.e, n shamir shares of msk using a degree $\lfloor \frac{n-1}{2} \rfloor$ polynomial. For all $i \in [n]$, set $\mathsf{pk}_i := [\mathsf{msk}_i]_2$. Output $\{\mathsf{msk}_i\}_{i\in[n]}, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{pk} := ([\tau]_1, \ldots, [\tau^B]_1, [\tau]_2, [\mathsf{msk}]_2)$.

- $\mathsf{Encrypt}(\mathsf{pk}, m, \mathsf{id}, t)$: Let $\mathbf{A}$ be a matrix in $(\mathbb{G}_2)^{2\times 3}$ and $\mathbf{b}$ be a vector in $(\mathbb{G}_T)^2$, defined as follows.

$$\mathbf{A} := \begin{pmatrix} [1]_2 & [\mathsf{id}]_2 - [\tau]_2 & 0 \\ [\mathsf{msk}]_2 & 0 & -[1]_2 \end{pmatrix}$$

$$\mathbf{b} := \begin{pmatrix} [0]_T \\ -([\mathsf{msk}]_2 \circ H(t)) \end{pmatrix}$$

  Sample a (column) vector $\mathbf{r} = (r_1, r_2) \leftarrow (\mathbb{Z}_p)^2$ and output the ciphertext c where

$$c = (\mathbf{r}^T \cdot \mathbf{A}, \mathbf{r}^T \cdot \mathbf{b} + m)$$

- $\mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\})$: Let $f(X) = \sum_{i=0}^{B} f_i \cdot X^i$ be a univariate polynomial of degree B over $\mathbb{Z}_p$ with roots at $\mathsf{id}_1, \ldots, \mathsf{id}_B$ and leading coefficient 1. Output digest $d := \sum_{i=0}^{B} f_i \cdot [\tau^i]_1$.

- $\mathsf{ComputeKeyShare}(\mathsf{msk}_i, d, t)$: Output the partial secret key $\mathsf{sk}_i := \mathsf{msk}_i \cdot (d + H(t))$.

- $\mathsf{ComputeKeyAggregate}(\{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in[n]}, d, t)$: Let $U = \{i | i \in [n], [1]_2 \circ \mathsf{sk}_i = \mathsf{pk}_i \circ (d + H(t))\}$. Let $V = \{v_1, \ldots, v_k\} \subseteq U$ be any arbitrary subset of U of size $k = \lfloor \frac{n-1}{2} \rfloor + 1$ and let $L_i(0) = \Pi_{j\neq i} \frac{(-v_j)}{(v_i - v_j)}$ be the i^{th} Lagrange coefficient for all $i \in [k]$. Output $\mathsf{sk} = L_1(0) \cdot \mathsf{sk}_{v_1} + \ldots + L_k(0) \cdot \mathsf{sk}_{v_k}$.

- $\mathsf{Decrypt}(c, \mathsf{sk}, d, \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}, \mathsf{id}, t)$: Let $q(X) = \sum_{i=0}^{B-1} q_i \cdot X^i$ be a univariate polynomial of degree $B-1$ with roots at $\{\mathsf{id}_1, \ldots, \mathsf{id}_B\} \setminus \{\mathsf{id}\}$ and leading coefficient 1. Set $\pi := \sum_{i=0}^{B-1} q_i \cdot [\tau^i]_1$ and set $\mathbf{w}$ to be the following vector.

$$\mathbf{w} = \begin{pmatrix} d \\ \pi \\ \mathsf{sk} \end{pmatrix}$$

  Finally, parse c as (c_1, c_2) and output the decrypted message $m^* := c_2 - c_1 \circ \mathbf{w}$.

Figure 2: Our construction for Thresholdizable Selective Batched IBE

We will now construct an adversary $\mathcal{B}$ which will contradict Corollary 5.2.1. The adversary $\mathcal{B}$ works as follows:

- If $|\mathsf{Cor}| < \left\lfloor \frac{n-1}{2} \right\rfloor$, let $\mathsf{Cor}' = \mathsf{Cor} \cup S$ where $S \subseteq [n] \setminus \mathsf{Cor}$ is an arbitrary subset s.t. $|S| = \left\lfloor \frac{n-1}{2} \right\rfloor - |\mathsf{Cor}|$.
- During the Setup and KeyGen phase, $\mathcal{B}$ performs the following steps:
  - Receive $(\mathsf{params}, \mathsf{pk})$ from the challenger.
  - For all $i \in \mathsf{Cor}'$, sample $\mathsf{msk}_i \leftarrow \mathbb{Z}_p$ and perform a $\mathbb{G}_2$ labeling query step using input msk_i to obtain $\mathsf{pk}_i = [\mathsf{msk}_i]_2$.
  - Let $U = \mathsf{Cor}' \cup \{0\}$. For all $u \in U$, let L_u be the lagrange polynomial of degree $\left\lfloor \frac{n-1}{2} \right\rfloor + 1$.
  - Set $\mathsf{pk}_0 := \mathsf{pk}$.
  - For all $i \in [n] \setminus \mathsf{Cor}'$, perform $\mathbb{G}_2$ group operation queries to obtain $\mathsf{pk}_i := \sum_{\forall u \in U} L_u(i) \cdot \mathsf{pk}_u$.
  - Send $(\mathsf{params}, \mathsf{pk}, \{\mathsf{msk}_i\}_{i \in \mathsf{Cor}}, \{\mathsf{pk}_i\}_{i \in [n]})$ to $\mathcal{A}$.
- During the *key computation queries* (during pre-challenge and post-challenge phases), $\mathcal{B}$ performs the following steps.
  - It receives a list ids of B identities along with a batch label t and forwards it to the challenger.
  - If the challenger halts the game, then $\mathcal{B}$ also halts.
  - Otherwise, $\mathcal{B}$ receives sk from the challenger and performs the following steps.
    - Compute a digest $d \leftarrow \mathsf{Digest}(\mathsf{pk}, \mathsf{ids})$ of the ids in ids using public key pk and $\mathbb{G}_1$ group operation queries.
    - Perform a hash query step to obtain $H(t)$.
    - For $i \in \mathsf{Cor}'$, perform a $\mathbb{G}_2$ group operation query to obtain $\mathsf{sk}_i := \mathsf{msk}_i \cdot (d + H(t))$.
    - Let $U = \mathsf{Cor}' \cup \{0\}$. For all $u \in U$, let L_u be the lagrange polynomial of degree $\left\lfloor \frac{n-1}{2} \right\rfloor + 1$.
    - Set $\mathsf{sk}_0 := \mathsf{sk}$.
    - For all $i \in [n] \setminus \mathsf{Cor}'$, use $\mathbb{G}_2$ group operation queries to get $\mathsf{sk}_i := \sum_{\forall u \in U} L_u(i) \cdot \mathsf{sk}_u$.
    - Send $\{\mathsf{sk}_i\}_{i \in [n] \setminus \mathsf{Cor}}$ to $\mathcal{A}$.
- During the challenge round, $\mathcal{B}$ forwards transparently forwards the messages received from $\mathcal{A}$ to the challenger and vice-versa.
- For all labeling queries, group operation queries, hash queries and pairing operation queries received from $\mathcal{A}$, forward it to the challenger, and then forward the response received back to $\mathcal{A}$.

By construction of $\mathcal{B}$, the following holds:

$$\Pr[\mathsf{Expt}_{\mathcal{B},0}^{\mathsf{SBIBE},\mathsf{GGM}}(1^\lambda, B) = 1] = \Pr[\mathsf{Expt}_{\mathcal{A},0}^{\mathsf{TSBIBE},\mathsf{GGM}}(1^\lambda, B, n, \mathsf{Cor}) = 1]$$

$$\Pr[\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}_{\mathcal{B},1}(1^\lambda, B) = 1] = \Pr[\mathsf{Expt}^{\mathsf{TSBIBE},\mathsf{GGM}}_{\mathcal{A},1}(1^\lambda, B, n, \mathsf{Cor}) = 1]$$

Moreover, the adversary $\mathcal{B}$ makes at most $q' = \mathsf{poly}(q, n) = \mathsf{poly}(\lambda)$ queries to the challenger. Hence, we get that,

$$\left|\Pr[\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}_{\mathcal{B},0}(1^\lambda, B) = 1] - \Pr[\mathsf{Expt}^{\mathsf{SBIBE},\mathsf{GGM}}_{\mathcal{B},1}(1^\lambda, B) = 1]\right| > \frac{1}{\mathsf{poly}_{\mathcal{A}}(\lambda)}$$

which contradicts Corollary 5.2.1. □