Integrating Group Signatures in Complex Decentralized Marketplace Transactions for Improved Buyer Privacy

Sen Qiao
Dept. of Computer Science
North Carolina State University
Raleigh, U.S.A
sqiao@ncsu.edu

Varun Madathil
Dept. of Computer Science
North Carolina State University
Raleigh, U.S.A
vrmadath@ncsu.edu

Kemafor Anyanwu
Dept. of Computer Science
North Carolina State University
Raleigh, U.S.A
kogan@ncsu.edu

***Abstract*—Marketplace applications are a popular application category for blockchains because they allow the possibility for parties to transact without the need of a trusted middleman entity. Supply chain marketplaces involve buyers and sellers of goods and services that interact using complex transactional workflows rather than single, one-off transactions. Further, in some contexts, there is a need to maintain some degree of privacy about which parties are transacting and/or the nature of their transactions. Keeping such information private is often crucial for maintaining competitive advantage. However, traditional public key signature schemes do provide strong enough privacy guarantees for such applications. On the other hand, signature schemes like *Group Signatures* offer more robust privacy guarantees but have not yet been adopted broadly in existing platforms. However, supporting privacy across multiple transactions in a workflow requires some extension to existing group signatures as opposed to just for a single transaction.**

In this paper, we present our effort on implementing some existing theoretical proposals for Group Signatures. We focus on application contexts in which it is sufficient to ”hide” the originator of a transaction workflow i.e., the buyer (but is easily generalizable). Because, transaction workflows involve ”multiple” related transactions which will be obfsucated by keeping introducing privacy, we extend a group signature scheme with ”linkability” of related transactions in a workflow. We integrate our extended signature scheme into SmartChainDB - our extension of an open source blockchain platform, BigChainDB, that introduces several features for enabling complex marketplaces on blockchains.

***Index Terms*—decentralized applications, blockchains, privacy, group signatures**

I. Introduction

Marketplaces are popular application category on blockchains are now known as *decentralized marketplaces*. Decentralized marketplaces address the major limitations of traditional marketplaces that rely on a trusted central entity to broker engagements between parties in a transaction. This centralization leads to problems of inefficiencies due to bottlenecks at the central entity and lack of transparency because central entity has full ownership and control of transactional data. Further, the central entity is at liberty to introduce arbitrary barriers to new market entry or to introduce other kinds of bias to their advantage. As an example, Amazon was accused of optimizing its proprietary search results ranking algorithm for prioritizing its profitability rather than showing the best results to customers [1].

In decentralized marketplaces, an application running on a blockchain undertakes the typical functions that a trusted centralized entity in a traditional marketplace does. In this context, execution of marketplace functions such as matching buyers with sellers and facilitating and recording transactions [2], are replaced by a network of nodes, each independently and concurrently accomplishing the same tasks, and have to come to a consensus about the outcome of transactions. There are many examples of emerging decentralized marketplaces. Some of them are simple involving one type of product and only one or two transaction types. For example, Lazooz [3] is a rideshare platform and similar to other ”single-function” marketplaces, only has a few transaction types: a ”request” (and payment) from a buyer and an ”accept” from a provider. However, marketplaces required to support supply chains can be quite complex involving different types of transactions for different phases. Those phases start from the procurement process for creating orders, include creating product to meet the orders and end with shipment and delivery. For example, IKEA which is the world’s largest furniture retailer sells its products in over 400 stores across 49 markets. Its supply chain and logistics ecosystem is a very complex network comprising about 1000 home furnishing suppliers across 51 countries, each of which has their own suppliers of e.g. product parts or materials [4]. In addition to the inefficiencies and high costs associated with traditional marketplaces, there are challenges with managing product traceability. Traceability is critical in identifying the appropriate scope for a product recall in the event that a product class has been identified as faulty. Further, transparency and traceability have also become a critical requirement for meeting the demands of customers to be given access to journeys of the products they consume - from origin to delivery. Consequently, the

* Work was done while author was a student at North Carolina State University

concept of decentralized supply chain marketplaces is gaining momentum. For example, IKEA [5] undertook a feasibility study for the use of permissioned blockchains to improve transparency and robustness of transaction processing in the context of a major international retail company.

One consequence of the complexity of supply chain marketplaces is that most transactions do not have isolated contexts but rather are often part of a transactional chain or workflow. For example, during the procurement phase, an entity e.g. a manufacturer, may want to find suppliers or subcontractors to produce a component. They will usually begin with issuing a ”request for quotes” (rfq) from suppliers who can then respond with quote bids. In many cases, there may be concern about sharing too many sensitive details such as product design as part of the rfq that is accessible to a large audience. Therefore, a preliminary ”pre-request” is usually used in the first step and then suppliers can respond with a ”letter of interest”. The requestor can perhaps then filter the list and narrow down to a smaller group of selected suppliers to whom the detailed request is then sent. Figure 1 illustrates this sequence of transactions in a two-sided marketplace.

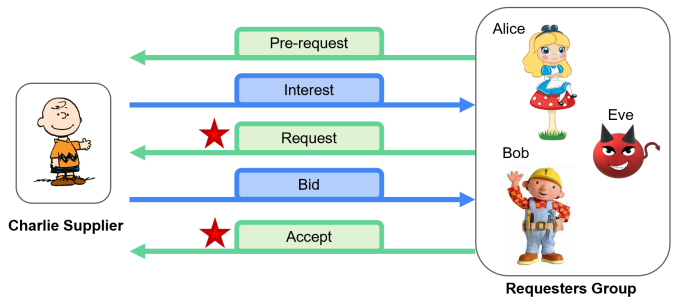

Fig. 1: Linking

One concern for transacting using shared ledger platforms like blockchains is privacy [6]. For example, for companies that are subject to GDPR and other privacy regulations, they must ensure that their applications meet necessary privacy requirements [7]. For market applications e.g. trading exchanges like private equity markets, brokers are required to maintain financial privacy of asset ownership and trades. This is a key requirement for equity trading markets [8], but is relevant to decentralized exchanges as well. The trend in supply chain processes is directed towards more and more collaboration and cooperation between manufacturers, distributors, and retailers. Because it is often necessary to exchange proprietary or sensitive information as a part of transactions in supply chain marketplaces, participants have become increasingly concerned about how this might negatively impact their competitive advantage [9]. Therefore, it may not be sufficient to just rely on traditional public key cryptography protocols because they are only able to provide ”pseudonymity” rather than absolute anonymity. Being able to exploit data on blockchains has led to negative consequences like front running attacks on trading exchanges [10] [11]. Further, [12] reports on the efforts to ”out” identities of individuals transacting on the BitCoin blockchain as part of criminal investigations. Such deanonymization is usually achieved through blockchain data mining and linking of external data such as IP addresses, wallet accounts to credit cards and so on.

Efforts are being directed at mitigating the privacy vulnerabilities in blockchains, particularly for applications where absolute privacy is critical. [13] addressed the issue of enhancing privacy in decentralized trading exchange marketplaces. For cryptocurrency platforms, approaches such as ”mixers” or ”coin-join” [14] techniques which mix, shuffle or lump together transactions so that it is not possible to specifically identify the endpoints of each transaction. However, such approaches are only useful for transactions involving fungible assets such as cryptocurrencies. Alternative strategies have been proposed to enable privacy-preserving cryptocurrencies like ZCash [15] which supports ”shielded transactions” that use privacy at the protocol level through zk-SNARKs as well as others [16] [17] [18]. Other efforts have proposed more advanced cryptographic signature schemes such as *Ring Signatures* and *Group Signatures*. These signature schemes provide individuals with signatures that are associated with a group of individuals rather than with the specific individual signing the signature. They do not use the asset in transactions as part of the scheme unlike the mixer approaches, therefore they can be applied more broadly to transactions with non-fungible assets. However, between the ring and group signatures, ring signature schemes do not offer non-repudiation which can be critical for supply chain applications. Therefore, group signatures might be the most appropriate for the supply chain contexts. Unfortunately, there is limited work in the area of group signatures for blockchain transactions. Further, there is still a need to address some of the unique challenges that arise in complex transaction scenarios like supply chain marketplaces. Specifically, given that supply chain marketplace transactions are often not independent but part of a chain or workflow: *how do we track related transactions by the same entity if users are anonymized*?

Example Scenario* - *Privacy and Linking Transactions: Recall our earlier example based on a two-sided supply chain marketplace in Figure 1. Now assume that the requestors, the originator of the workflows (on the right side of figure) want to be anonymous. If a group signature scheme is being used, requestors will be placed in groups and given group keys with which to create a group signature when signing a transaction. The signature allows the transaction to be linked to ”a member of the group” but not to a specific individual. Suppose `Alice` was the originator of the transaction chain i.e. she initiated the `PreRequest` transaction for suppliers that may be interested in engaging in a supply contract. Subsequently, a `Request` transaction will follow and then an `Accept` transaction (after suppliers have submitted `Bid` transactions. However, the `PreRequest`, `Request` and `Accept` were all signed with group signatures merely identifying the group but not the individual. Without being able to resolve the actual identities of signers, it is possible for adversaries to ”attack” the `Request` and `Accept` transaction steps, submitting false transactions, without the transaction recipient (`Charlie`) realizing. For example, the adversarial actor could initiate the `Accept` transaction, selecting the worst of all the bids.

The example scenario demonstrates the need to be able to "link" related transactions i.e. all transactions in the workflow from the same entity, even though entities are not explicitly identified like with group signatures. In our example, we have two classes of related transactions: the green transaction chain for the requestors and the blue transaction chain for the responders or suppliers. While our example discussion focused on only the requestor side being anonymous, it is not hard to anticipate scenarios where the blue class (suppliers) also want to be anonymous and will need to use group signatures as well. Therefore, some consideration for how to achieve "linkability" of different transactions in a chain while using group signatures for anonymity is necessary.

In this paper, we will build on existing work on group signatures [19] [20], adding linkability and integrating in the transaction architecture of a blockchain platform which we here call **SmartChainDB**. **SmartChainDB** is an extension of the open source blockchain database BigChainDB that aims to provide many of the common marketplace transactional functions as first-class transactions, eliminating the need to implement such functions as smart contracts. BigChainDB has as its foundation a scalable, distributed NoSQL database, on top of which blockchain functionality is added. Specifically, we

1) motivate the need for privacy in complex marketplace workflows
2) propose implementation framework that builds on existing work on group signatures to enable linkability of multiple related transactions under anonymity
3) demonstrate how it can be integrated into a blockchain platform and report experimental results

In the rest of the paper, we discuss background and related work, approach and evaluation in sections II, III and IV respectively.

II. Background and Related Work

A. *Overview of* SmartChainDB

SmartChainDB introduces a set of new transaction types such as `Request`, `PreRequest`, `Bid`, `Interest` and a few others, into its foundational component - BigChainDB - an open source blockchain database that provides a good hybrid of blockchain and scale-out database properties. The aim of **SmartChainDB** is to provide native support for such marketplace transaction types including system-based validation, so that users can use them out-of-the-box and not be burdened with always implementing such behavior as smart contracts. In effect, **SmartChainDB** users can initiate such transactions much like they would a typical `Transfer` transaction without having to worry about implementing code to check semantic errors such as doublespend.

BigChainDB provides blockchain properties such as decentralization, Byzantine fault tolerance, owner-controlled assets, and immutability ensure various asset transactions are in a decentralized system can be saved forever. Simultaneously, it offers the benefits of scale-out databases such as high transaction rate, low latency, and indexing and querying of structured data, ensure transaction speed and overall usability. Its layered architecture comprises a storage layer which uses `MongoDB`; a consensus layer which uses `Tendermint` [21], [22] to provide asynchronous byzantine fault tolerance; a server layer which implements the semantic validation of transactions; and a client layer which has a driver used to interact with the system. BigChainDB uses a declarative transaction model in which transactions have schemas so that introducing new transaction types amounts to defining corresponding transaction schemas as well as extending the server validation code to support validation of the new transaction types.

With respect to identity management for transactions, BigChainDB uses the Ed25519 public-key signature system for: generating public/private (verifying/signing) key pairs, calculating and verifying Ed25519 signatures. Digital signatures can be considered as the digital counterparts of stamped seals or handwritten signatures. They allow a recipient of a message to verify that a message has from the indicated sender. In general, a digital signature involves three steps: (i.) computing the hash value of message to be sent; (ii.) signing the hashing value with a private key and attaching the signature to the message before sending; (iii.) the recipient can verify by hashing the original message and using the sender's public to sign which should produce the same output if valid. Ed25519 is an instance of the Edwards-curve Digital Signature Algorithm (EdDSA).

Similar to the BitCoin transaction model, keys and signatures are attached to transaction *inputs* and *outputs* in BigChainDB. A BigChainDB transaction has a semi-structured object payload with some standard fields. Figure 2 shows a diagrammatic representation of a transaction. The *asset* and *metadata* fields store key data (immutable) or metadata (mutable) about the asset being used in a transaction e.g. for a property its address may be part of the *asset* field whereas the color of the property will be in its *metadata* since that can be changed in a future transaction. In this model, any kind of asset can be used in a transaction once the definition of the correct transaction structure is captured by a transaction schema. The *operation* identifies the kind of transaction it is, the *id* field is a transaction id that is generated as a hash of the rest of the transaction payload except the *id* field. For the purposes of our discussion we will focus primarily on the *inputs* and *outputs*.

An output *condition* (following the crypto-conditions specification) locks a transaction, such that only a valid *input fulfillment* can unlock it. "In the case of signature-based schemes, the lock is basically a public key, such that in order to unlock the transaction one needs to have the private key". For example, assume that transaction $\texttt{T}_1$ is a `CREATE` transaction which creates an asset on the blockchain. The conditions in the output of such a transaction would have the public keys of the entities authorized to spend or consume the associated asset.

This would imply that any subsequent transaction e.g. $\mathtt{T}_2$ that has an input trying to consume or spend $\mathtt{T}_1$'s output must have a signature that was generated based on the the private key associated with the public key given in the output conditions. Using a public-key based signature system, it is easy to link multiple transactions belonging to the same owner because their keys are known. However, as mentioned earlier, public key cryptography does not offer adequate levels of privacy. Figure 2 illustrates the linking between two transactions. The relevant details transaction model and blockchain platform are summarized in later sections. However, a completely detailed discussion of the transaction model and transaction processing is outside of the scope of the discussion in this paper and are omitted.

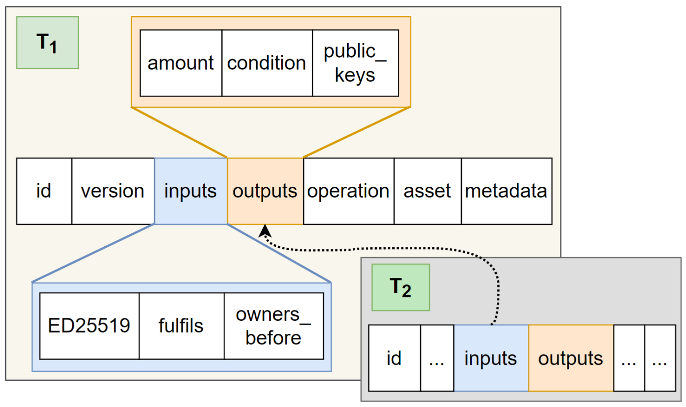

Fig. 2: Transactions

B. Privacy-Preserving Transaction Schemes

Some existing tools aim to achieve user anonymity via a mixer approach. Decentralized mixers such as CoinShuffle (or Coinparty, XIM, CoinShuffle++) [23] will combine multiple transactions into a huge CoinShuffle transaction that can trace back anyone that has participated in it. Thus, transactions from the sender cannot be linked to the transactions to the receiver, assuming there are multiple unique senders and receivers. However, this approach is only appropriate where there is only one transaction type so that all instances of such a transaction can be lumped together so that they become indistinguishable. Consequently, it is not adequate to address our problem with transaction chains containing different transactions types. Alternative proposals have used a ”crowd approach” which uses an anonymity set to mask users so that they become indistinguishable making it difficult to link data back to the users. Increasing the size of the anonymity set provides better anonymity guarantees. *Ring Signatures* and *Group Signatures* use this crowd approach. In cryptography, a *Ring Signature* is a type of digital signature where each member of the ring has keys and any such member can sign a message (with contributions with other members of the ring) to form a ring signature. This signature only indicates that *some* member of the ring has endorsed the message, but doesn't specify which one has. Anyone with the public keys can verify the incoming signature from that particular ring. Ring signature can provide linkability [24] [25] using the same list of public keys of all ring members. However, given that a property of ring signatures is computationally infeasible to determine which of the group members' keys were used to produce the signature, it is impossible to revoke the anonymity of a ring signature. This has practical limitations for applications like supply chain marketplaces because it is important to be able to ”out” rogue participants and to take legal actions where necessary. The set of members to form the ring must be decided a priori because adding additional members changes the set of public keys that comprise the ring and renders any earlier signatures based on the old ring set unverifiable. Furthermore, scalability of ring signature functions degrades as the ring size increases.

1) Group Signatures: Group signatures use a similar idea of anonymity of an individual within a group but add the possibility of non-repudiation of an individual. A key difference in the structure is the existence of a group manager who is trusted to administer and maintain information about groups and their members. This manager has the capability to reveal a group member's identity if the appropriate group signature method is implemented. Earliest proposals in this category were RSA-based Short Group Signature schemes [26] [27] and the more efficient bilinear maps based group signature schemes [28] [29]. Recently, Bichsel & Camenisch [19] proposed a fast bilinear mapping and re-randomization-based group signature scheme. Pointcheval & Sanders [20] propose an efficient group signature scheme implementation scheme based on [19]. Some recent efforts have considered the issue of ”linkability” of group signatures in asynchronous blockchain networks. [30] presents linkable group signature scheme for payers of cryptocurrencies using linear encryption group managers and enables the prevention of double-spending and uses the `TRANSFER` amount as a parameter. However, it is only applicable to linking transactions of the same type not to the workflows of different transaction types as it is being considered in this paper.

Discussion: For our application context that requires both transaction owner privacy and non-repudiation, using group signatures is the most suitable. We further require the connections between different transactions within a transaction workflow, but the same anonymous owner, are not lost under the anonymity provided by such privacy. In the rest of the paper, we will present **SmartChainDB**'s approach for transaction user privacy based on an implementation of the scheme proposed in [20] that is also extended to support ”linkability”.

III. Implementation Framework for Group Signatures in SmartChainDB

A. Approach Overview

The overall implementation objective is threefold: (i.) realize a concrete implementation of the earlier mentioned group signature scheme proposed in [20]; (ii.) extend that scheme to supporting linkability under ownership privacy (iii.) integrate this signature scheme implementation into the

SmartChainDB blockchain transaction processing architecture in order to enable support for users who need to sign transactions anonymously due to application requirements.

There are 4 key entities in [20]'s group signature scheme: the $groupmanager$ - is a trusted entity responsible for creating and managing all groups and their signatures; the *user* - a user member of some group; the *certification authority* (CA) - providing users with support for traditional public/private key cryptography which is used as a building block for their group signatures; and the *verifier* - anyone who receives a message signed with a group signature and wishes to verify the signature. The scheme is comprised of a set of the key functions: `GSetup`(), `PKIJoin`(), `GJoin`(), `GSign`(), `GVerify`().

Figure 3 shows the functions and parties involved in the user and group initialization steps, as well as, the joining of a group by a user that provides them with a group signature for the joined group. To give a more complete illustration, the figure also shows the subsequent steps in the group signature workflow which include signing the first message/transaction (e.g. the `PreRequest` transaction) with the group signature, and then the recipient (here the blockchain validator) verifying the signature.

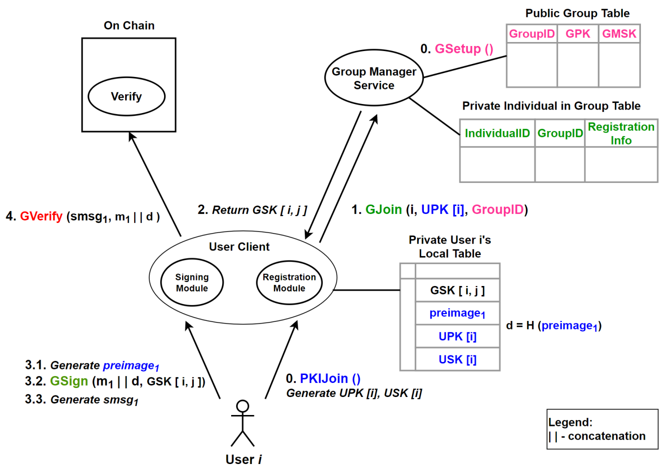

Fig. 3: Workflow process with different preimages

We elucidate the steps in the group signature scheme in more detail and use the transactions in our proposed marketplace workflow such as `PreRequest` as examples where appropriate.

1) Step 0 (`GSetup` and `PKIJoin`): initializes any new groups being created or any new user joining the system. For each new group, the $groupmanager$ runs `GSetup` to generate its *group public key* (GPK) and a *group master secret key* ($GMSK$) and assigns it a $GroupID$. GPK is made publicly available to be used by anyone who wants to verify group signatures signed by members of that group while $GMSK$ is kept only by the $groupmanager$. The generation of ($GMSK$) is parameterized by a security input parameter 1^η which corresponds to key length (the longer the key the more secure). A `user` i who wants to join some group must first use the method `PKIJoin` that runs a DSKeyGen (digital signature generator) to generate a public-private key pair which afterward, is stored and managed by the CA (*certification authority*). The public key is denoted $UPK[i]$ while the private or secret key is denoted as $USK[i]$.
2) Steps 1 & 2 (`GJoin`): in 1, a `User` i makes a request to the $groupmanager$ to join a group j using its `GJoin` function. `GJoin` uses the outputs of the `PKIJoin` & `GSetup` functions: $(USK[i], UPK[i], GMSK, GPK_j)$ to generate $GSK[i, j]$ ($user$ i group secret key for group j) maintained by $user$ i and $Reg[i-j]$ (registration information) maintained by the $groupmanager$. $GSK[i-j]$ will allow users to sign their messages, while $Reg[i, j]$ will allow Group Managers to identify a message's true signer and not just the group that the signer belongs to. The figure shows the table structures used by both parties to maintain this information. In step 2 of the flow, $GSK[i, j]$ is returned to $user$ i.
3) Step 3 (`GSign`()): In this step, $user$ i creates and uses the `GSign`() method and their assigned group signature secret key $GSK[i, j]$ to sign a transaction (in this example a `PreRequest` transaction). `GSign`() takes two parameters: the message to be signed and the secret key used for the signing ($GSK[i, j]$) and then outputs the signed message. The message to be signed is a created as a concatenation of the transaction payload (denoted as m in figure) and a value d $(m||d)$. d is the hash of some randomly generated value (denoted as preimage$_1$). The result is the signed message $smsg_1$.
4) Step 4 (`GVerify`()): In this step, a recipient of the signed message can verify that the signature is from a valid group member by running `GVerify`() with three parameters: the signed message $smsg_1$, the raw message unsigned $(m||d)$ and the public key for the group from which the group member is GPK_j.

For each new transaction that the user wants to sign and submit to the blockchain, Steps 3 and 4 need to be repeated but Steps 1 and 2 are only done once. In the above example, we will assume that `user` i is a $Requestor$ who is initiating a supply chain flow by submitting a first request using the `PreRequest` transaction. This means that the transaction payload that formed the message m in Step 3 is a `PreRequest` transaction payload. Now suppose that subsequently, `user` i wants to move the process along after receiving some `Interest` transactions from different suppliers. Now, `user` i or $Requestor$ continues the process by submitting a `Request` transaction. In this case only Steps 3 and 4 are needed.

B. Group Signature Implementation

As of the time of this work, we could not identify a publicly available implementation of the [20] group signature scheme. We, therefore, implemented the signature scheme, adding necessary extensions and modifications to suit our context. Although our target implementation context (i.e. the **SmartChainDB** platform) is a python implementation, we opted to implement the [20] group signature framework in Rust in order to leverage some publicly available implementation building blocks and, therefore, to avoid re-implementing all building blocks from scratch in python. The most suitable foundational implementation framework found was Hyperledger Ursa's cryptographic library. Hyperledger is an open-source, high credibility, and high-quality repository for blockchain and related technologies. Hyperledger Ursa is a Rust-based library that provides some key building blocks such as *Simple Digital Signatures*, *Sigma Protocols*, and the *bilinear mapping* and *re-randomization* schemes proposed by Pointcheval-Sanders [20]. The Ursa library also contains some of the functions from [19] upon which [20] builds such as: `Setup`, `KeyGen`, `DSSign`, and `DSVerify`. More specifically, Hyperledger Ursa has a Short Randomizable signatures folder which is based on the Pointcheval & Sanders paper [20]. This folder contains both simple digital signature signing and proof of knowledge functions, which form the necessary backbones of `GSign` and `GVerify` based on the Bichsel and Camenisch scheme [19]. Further, since the Rust library borrows the C Apache Milagro Cryptographic Library, we anticipated that it would likely be more efficient option than a Python implementation. We implemented Python wrappers to call the Rust cryptographic functions as services. We implemented a Cherry.py server to take the role of *groupmanager* as a service. The pseudocode for the key methods in the scheme is given in the appendix. A few are omitted due to space constraints.

The [20] group signature scheme specifies additional functions GOpen() - that can be used by the GroupManager to identify the signer's actual identity using the registration information $Reg[i]$ in the event that there is need to reveal it and GJudge() can be used in the case that an adversary comprises the GroupManager to check if the user in question is the correct user. However, our current implementation omits these two functions to minimize the risk of privacy leaks. Integrating group signatures into **SmartChainDB** required (i.) modification o f t ransaction m odel s hown in Figure 2 and (ii.) introducing new system components such as the **GroupManagerService** as well as modification of some of the existing system components such as the BlockchainServer and the Driver to support processing.

1) System Architecture: Figure 4 shows the key components of **SmartChainDB** and a high level depiction of the interaction flow u sed f or t ransaction p rocessing. I t highlights the three main subsystems: the **GroupManagerService**, the **UserClient** and a single **BlockchainNode**. The **GroupManagerService** includes GroupManager code that implements management methods like GJoin and its local database. The **Client** includes the Driver code used to interact with the blockchain and the programming interface for interacting with the **GroupManagerService** that offers functions like GSign or GVerify which the users can use signing transactions or verifying signatures respectively. Each **BlockchainNode** comprises the BlockchainServer which supports the validation of transactions based on appropriate transaction semantics e.g. rejecting doublespending for

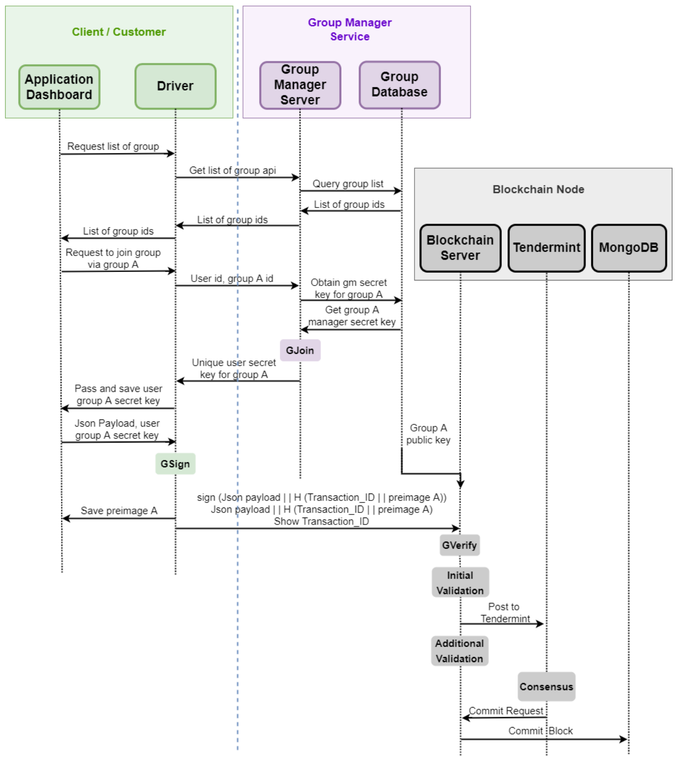

Fig. 4: System Components & Transaction Processing Flow

TRANSFER transactions. Each **BlockchainNode** also contains Tendermint for consensus and MongoDB for storage of the blockchain. The BlockchainServer validates transactions in multiple stages, but first the signatures are validated. The Server includes a wrapper for the function GVerify so that if a transaction specifies the use of the group signature scheme, the GVerify function is called to verify the signature rather than the default ED2259 signature verifier. (Subsequent validation stages validate other transaction semantics and then send to the consensus phase which also does some additional transaction validation before finally posting in database. We ignore the details of transaction processing here).

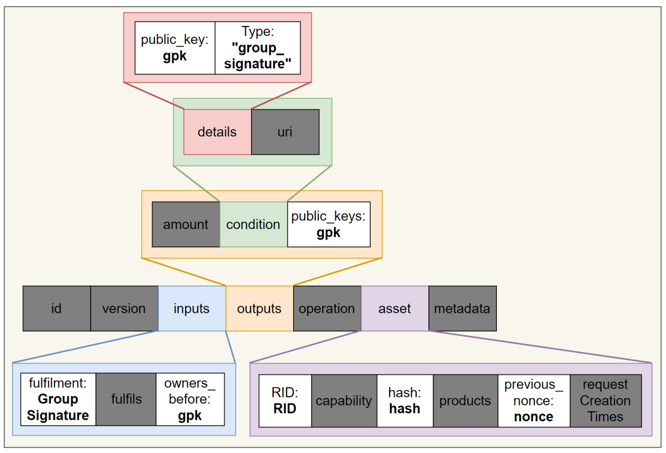

Fig. 5: Modified Transaction Structure

SmartChainDB's transaction model also had to be modified from what is shown in Figure 2 to Figure 5. The modified model introduces the use of reserved keywords in the transaction schema for indicating the selection of "group signature" as the signature type for a transaction. Then, it permits the use of group public keys and group signatures in transaction input and output fields for signing transactions if the "group signature" type is indicated (the older model permitted Ed22519).

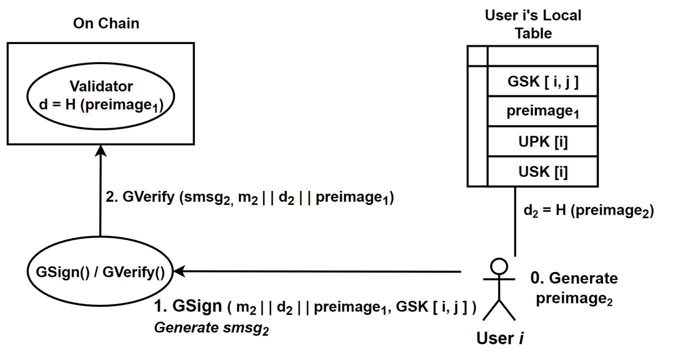

Fig. 6: Workflow for a Subsequent Transaction after `PreRequest`

2) Linking Related Transactions Signed Using Group Signatures: We integrate the use of form of hashes and preimages for linkability to ensure that we can link the transactions made by the same group member in spite of their identities are private. Figure 6 illustrates this process for the *second* transaction in the marketplace transaction chain being submitted by `user` i i.e. the `Request` transaction. Submitting this transaction requires the generation of another preimage (random number) preimage$_2$, that will be used to create the message to be signed. However, in this case, not only does the message include m_2 - the `Request` transaction payload and the hash of preimage$_2$ which is d_2, it also includes preimage$_1$ (m_2 || d_2 || preimage$_1$). This is signed to produce $smsg_2$. Since only `user` i is privy to the preimage included in the previous step, only `user` i can generate the correct message in the second step which includes $preimage_1$. This can be verified by the **Verifier**. In this way, we allow one transaction to be linked to another only if the user can provide a valid preceding preimage. The extended transaction model in Figure 5 also shows the additional data used for linkability i.e. the preimages, denoted as "nonce" in the figure.

IV. Evaluation

Experiment Setup: Ubuntu version 18.04 virtual machine, Docker version 20.10.5, Python version 3.6.9, Cargo version 1.53.0, and `MongoDB` Compass version 1.26.1. We set Group Manager web service to port 8080, BigchainDB Server access port to default (9984), and `MongoDB` port to default (27017). To assess the practicability of using the "linkable" group signature for blockchain transactions, we ran some evaluation experiments to see how much of an additional overhead such as scheme adds to transaction processing. (Given the complexity of the signature scheme some overhead is to be expected). We conducted three experiments: (i.) comparison of time costs for signing and verifying group signatures vs. the existing ED25519 public-key signature; (ii.) scalability of the group signature with increasing group sizes; (iii.) comparison of transaction processing latencies when using group signatures vs. using ED25519 public-key signatures.

Comments: In interpreting results, we must note that the two schemes do not offer the same features: the ed25519 signature scheme does not offer the anonymity (linkability is a non-issue here). Further, unlike the homogeneous runtime environment of ed25519 signature scheme, the programming environment of the group signature is more complex involving Rust, C, Python and Java runtimes which adds some runtime overhead independent of the group signature scheme.

i. **Signing Costs Comparison**: For this experiment, we ran 200 times for each of the following functions: the ed25519 sign, ed25519 verify, group signature sign, and group signature verify, all signing and verifying the same message.

TABLE I: Time Cost of Group Signature

Average Time Cost		
Type of function	Std	Mean
Signing Digital Signature - Python (Ed25519)	0.8402744531	0.748246195ms
Verifying Digital Signature - Python (Ed25519)	0.6407945735	1.006927615ms
Signing Group Signature - RUST	0.9230209381	9.772752225ms
Verifying Group Signature - RUST	0.298527082	11.12937439ms

Table I reports average time cost (from 200 runs) for each case. We see that although there is an additional overhead of the signing process for group signatures when compared to the traditional ed25519 signature (as expected), in practical terms, this translates to less than 10ms which is not very significant. Further, some of the increase can also be attributed to the more complex runtime architectures of this specific group signature implementation that built on existing building blocks developed in different programming runtimes.

ii. **Scalability With Increasing Group Size**: This experiments evaluates how group signature performance is impacted as the number of group members (i.e. the anonymity set size) increases. Here we compared the average time (from 200 runs) it takes for group signing and group verifying for 2, 4, 8, 16 group members.

TABLE II: Scalability of Group Signature

Group Growth		
Num of members	Types of function	Group Signature Time Cost
2	Signing Verifying	4.081926ms 4.667232ms
4	Signing Verifying	4.062314ms 4.366295ms
8	Signing Verifying	4.264439ms 4.578303ms
16	Signing Verifying	4.501189ms 4.341704ms

We see from Table II that the group signature performance stays relatively stable with increasing group sizes. This is an advantage of group signatures which signing process is very dependent on the members in the ring and therefore negatively impacted by ring sizes (well known property, not evaluated here).

iii. **Comparison of Transaction Processing Performance**: Lastly, we are interested in the effects our implementation has on the overall marketplace transaction workflow. In this experiment (Table III), we ran 5 transactions `PreRequest`, `Interest`, `Request`, `Bid`, and `Accept`, where we've added both linkable nonces and group signature to replace the elliptical curve group signature.

From Table III, we see our group signature overhead on transactions on average adds 23347 ms (377%) increase.

TABLE III: Comparing Transaction Times

Transaction Times		
Trial (ms)	Ed25519	Group Signature
Transaction 1	12096.54662ms	31111.50953ms
Transaction 2	5829.050265ms	21922.53284ms
Transaction 3	6111.45259ms	21595.47869ms
Transaction 4	5989.259657ms	19268.20824ms
Transaction 5	5883.072631ms	34866.42132ms
Transaction 6	5949.524186ms	38999.45385ms
Transaction 7	5711.611076ms	39740.79425ms
Transaction 8	6097.671397ms	32362.10934ms
Transaction 9	5807.926217ms	39448.26101ms
Transaction 10	6017.69875ms	36020.34113ms
Transaction 11	5978.015848ms	37758.13883ms
Transaction 12	5759.222706ms	35866.61027ms
Transaction 13	5630.697453ms	33903.15254ms
Transaction 14	5671.375385ms	15219.04399ms
Transaction 15	5745.127919ms	29837.69522ms
Transaction 16	5699.859339ms	39040.97876ms
Transaction 17	6015.530188ms	21387.40648ms
Transaction 18	5977.222206ms	16074.89105ms
Transaction 19	6062.802051ms	15646.88341ms
Transaction 20	5680.885235ms	30587.86738ms

To summarize the reasons for these increased costs, we acknowledge that it is expected that the addition of hash verification, group signing, group verifying, and groups and language switching does add some overhead to transaction processing. Some of the overhead is justified by the fact that this scheme adds two group signature calls at driver and two group signature verification steps at the server within each transaction signed with the new group signature transactions. In addition, we introduced two hash function steps at the Driver layer and two hash verification steps by the Server.

TABLE IV: Statistical Analysis of TABLE III

Statistical Value		
	Ed25519	Group Signature
Average:	6185.727586ms	29532.88891ms
Std (σ):	1399.817622	8775.26162
Coefficient of variation (CV):	0.2262979743	0.297135226
95% Confidence Interval:	6185.727586± 655.1348134ms	29532.88891± 4106.948858ms

However, we believe the main source of the increase in time costs is merely a function of implementation. A key bottleneck would be transferring the signatures and public keys via shellcode, which heavily depends on the system and not the effectiveness of the code. In contrast, in the original ed25519, the (Python) BigchainDB driver signs using a native python cryptoconditions library.

In the ed25519 verifying process, the (Java) BigchainDB server verifies using a native java ed25519-java library. Both do not need to call another language at all. Lastly, we added four language switches (of which two calls are Python to Rust to C++ and two calls are from Java to Rust to C++). Assuming hash creation and hash verification are negligible (as the hash function used is built-in). Given the results from experiment (i.) that tested the signature schemes in standalone mode (not integrated with transaction processing), we hypothesize that a significant proportion of this latency here is due the multiple switches in programming language runtimes. Consequently, a reimplementation of the group signature code in the same runtime as the blockchain platform should significantly improve performance.

V. Conclusion

In this paper, we present an approach for supporting privacy in the context of complex marketplace transactional workflows using group signatures and an extension to support linkability of different transactions types. We developed an implementation and integrated it into the transaction processing workflow of a blockchain platform being developed on top of BigChainDB. The results showed some expected additional overhead due to the new, more complex signature scheme. However, we also observed that a significant overhead was introduced due to the runtime architecture chosen for the implementation reported here which is not a requirement but was a choice for ease of implementation and can be replaced to improve performance.

VI. Acknowledgements

This work was partially funded by the National Science Foundation CSR grant #1764025. We would also like to thank Drs. Alessandra Scafuro and Binil Starly for their contributions in terms of very useful discussions and feedback.

References

[1] D. Mattioli, "Amazon changed search algorithm in ways that boost its own products," Wall Street Journal, 2019.
[2] H. Subramanian, "Decentralized blockchain-based electronic marketplaces," Communications of the ACM, vol. 61, no. 1, pp. 78–84, 2017.
[3] "La'zooz white paper," 2015. [Online]. Available: https://tinyurl.com/LazoozWhitePaper
[4] T. Sund, C. Lööf, S. Nadjm-Tehrani, and M. Asplund, "Blockchain-based event processing in supply chains—a case study at ikea," Robotics and Computer-Integrated Manufacturing, vol. 65, p. 101971, 2020.
[5] T. Sund, C. Lööf, S. Nadjm-Tehrani, and M. Asplund, "Blockchain-based event processing in supply chains—a case study at ikea," Robotics and Computer-Integrated Manufacturing, vol. 65, p. 101971, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584519301905
[6] "The privacy questions raised by blockchain," Bradley LLP, January 14, 2019 [Online]. [Online]. Available: https://www.coincenter.org/education/advanced-topics/what-are-mixers-and-privacy-coins/
[7] E. Natalie and T. Jansen, "Blockchain and data protection law: when anonymous data becomes personal," dotmagazine, 2017.

[8] , “Nasdaq Private Market,” 2022, [Online]. [Online]. Available: https://www.nasdaq.com/solutions/private-company-solutions
[9] Y. Hong, J. Vaidya, and S. Wang, “A survey of privacy-aware supply chain collaboration: From theory to applications,” Journal of Information Systems, vol. 28, no. 1, pp. 243–268, 2014.
[10] S. Eskandari, S. Moosavi, and J. Clark, “Sok: Transparent dishonesty: front-running attacks on blockchain,” in International Conference on Financial Cryptography and Data Security. Springer, 2019, pp. 170–189.
[11] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, “Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges,” arXiv preprint arXiv:1904.05234, 2019.
[12] L. Andrew and A. Douglas, “Bitcoin investigations: Evolving methodologies and case studies,” J Forensic Res, vol. 9, p. 420, 2018.
[13] K. Govindarajan, D. Vinayagamurthy, P. Jayachandran, and C. Rebeiro, “Privacy-preserving decentralized exchange marketplaces,” arXiv preprint arXiv:2111.15259, 2021.
[14] A. V. WIRDUM, “Coinjoin’s first steps: How dark wallet paved the way for a more private bitcoin,” Bitcoin Magazine, February 25, 2020 [Online]. [Online]. Available: https://bitcoinmagazine.com/culture/coinjoins-first-steps-how-dark-wallet-paved-the-way-for-a-more-private-bi
[15] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, “Zerocash: Decentralized anonymous payments from bitcoin,” in 2014 IEEE symposium on security and privacy. IEEE, 2014, pp. 459–474.
[16] N. Narula, W. Vasquez, and M. Virza, “{zkLedger}:{Privacy-Preserving} auditing for distributed ledgers,” in 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 2018, pp. 65–80.
[17] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi, “Solidus: Confidential distributed ledger transactions via pvorm,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 701–717.
[18] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, “Zether: Towards privacy in a smart contract world,” in International Conference on Financial Cryptography and Data Security. Springer, 2020, pp. 423–443.
[19] P. Bichsel, J. Camenisch, G. Neven, N. Smart, and B. Warinschi, “Get shorty via group signatures without encryption,” in Security and Cryptography for Networks - SCN 2010, vol. 6280. Germany: Springer Berlin Heidelberg, 2010, pp. 381–398, other page information: 381-398 Conference Proceedings/Title of Journal: Security and Cryptography for Networks - SCN 2010 Other identifier: 2001254.
[20] D. Pointcheval and O. Sanders, “Short randomizable signatures,” 2015, https://eprint.iacr.org/2015/525.
[21] E. Buchman, “Tendermint: Byzantine fault tolerance in the age of blockchains,” 2016.
[22] J. Kwon, “Tendermint : Consensus without mining,” 2014.
[23] J. Bernal Bernabe, J. L. Canovas, J. L. Hernandez-Ramos, R. Torres Moreno, and A. Skarmeta, “Privacy-preserving solutions for blockchain: Review and challenges,” IEEE Access, vol. 7, pp. 164 908–164 940, 2019.
[24] J. K. Liu1 and D. S. Wong, “Linkable ring signatures: Security models and new schemes,” in Computational Science and Its Applications – ICCSA 2005. Berlin, Heidelberg: Springer, 2005, pp. 614–623. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2013.34
[25] B. Goodell, S. Noether, and RandomRun, “Concise linkable ring signatures and forgery against adversarial keys,” 2019, https://eprint.iacr.org/2019/654.
[26] D. Boneh and H. Shacham, “Group signatures with verifier-local revocation,” in CCS ’04, 2004.
[27] A. Ishida, Y. Sakai, K. Emura, G. Hanaoka, and K. Tanaka, “Proper usage of the group signature scheme in iso/iec 20008-2,” in Asia CCS ’19, 2019.
[28] J. Camenisch and A. Lysyanskaya, “Signature schemes and anonymous credentials from bilinear maps,” in CRYPTO, 2004.
[29] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, “Aggregate and verifiably encrypted signatures from bilinear maps,” IACR Cryptology ePrint Archive, vol. 2002, p. 175, 2002.
[30] L. Zhang, H. Li, Y. Li, Y. Zhao, and Y. Yu, “An efficient linkable group signature for payer tracing in anonymous cryptocurrencies,” Future Gener. Comput. Syst., vol. 101, pp. 29–38, 2019.

VII. Appendix

GSetup(1^η):

Inputs: 1^η
Output: $GPK, GMSK$

1) Group Manager: $x \leftarrow Z_p, y \leftarrow Z_p$
2) Group Manager: $X_2 \leftarrow g_2^x\ Y_2 \leftarrow g_2^y$
3) Group Manager: $GPK \leftarrow (g_2, X_2, Y_2)$, $GMSK \leftarrow (x, y)$

PKIJoin(i):

Inputs: i
Output: $USK[i], UPK[i]$

1) User and CA: $(UPK[i], USK[i]) \leftarrow DSKeyGen()$

GJoin($i, USK[i], UPK[i], GMSK, GPK$):

Inputs: $i, USK[i], UPK[i], GMSK, GPK$
Output: $GSK[i], Reg[i]$

1) User: generates $ski \leftarrow Z_p$, $(\tau, \tau_2) \leftarrow (g^{ski}, Y_2^{ski})$, $\eta \leftarrow sign(USK[i], \tau)$
2) User: sends (τ, τ_2, η) to Group Manager
3) Group Manager: upon receiving (τ, τ_2, η), verifies that $\hat{e}(\tau, Y_2) = \hat{e}(g, \tau_2)$ and verifies η with τ and $UPK[i]$(received from Cerification Authority). Executes a Sigma Protocol with User showing that User knows ski. Then Group Manager generates $u \leftarrow Z_p$ and create $\sigma \leftarrow (\sigma_1, \sigma_2) \leftarrow (g^u, (g^x \cdot (\tau)^y)^u)$. Lastly stores $[i, \tau, \eta, \tau_2]$ as secret register
4) Group Manager: sends (σ) to the User
5) User: upon receiving (σ), User saves $(ski, \sigma, \hat{e}(\sigma_1, Y_2))$ as $GSK[i]$

GSign($GSK[i], m$):

Inputs: $(ski, \sigma, \hat{e}(\sigma_1, Y_2)), m$
Output: μ, m

1) User: Re-randomization - generate $t \leftarrow Z_p$. Then caluates $(\sigma_1', \sigma_2') \leftarrow (\sigma_1^t, \sigma_2^t)$
2) User: Computes Signature Proof of Knowledge (Σ) for ski see Figure **??**. User first calculates $k \leftarrow Z_p$, computes $A = \hat{e}(\sigma_1', Y_2)^k = \hat{e}(\sigma_1, Y_2)^{k \cdot t}$, then compute $c = H(\sigma_1', \sigma_2', \hat{e}(\sigma_1, Y_2)^{k \cdot t}, m)$, and then computes $s \leftarrow k + c \cdot ski$
3) User: Lastly outputs μ and m, where $\mu \leftarrow (\sigma_1', \sigma_2', c, s)$ and where c is the challenge and s is the response.

GVerify(GPK, μ, m):

Inputs: GPK, μ, m where $\mu \leftarrow (\sigma'_1, \sigma'_2, c, s)$ and $GPK \leftarrow (g_2, X_2, Y_2)$
Output: $0/1$

1) Verifier: Compute $R \leftarrow (\hat{e}(\sigma_1'^{-1}, X_2) \cdot \hat{e}(\sigma'_2, g_2))^{-c} \cdot \hat{e}(\sigma_1'^{s}, Y_2)$
2) Verifier: check $c = H(\sigma'_1, \sigma'_2, R, s)$ to verify the signature of knowledge, this works since $(\hat{e}(\sigma_1'^{-1}, X_2) \cdot \hat{e}(\sigma'_2, g_2))^{-c} \cdot \hat{e}(\sigma_1'^{s}, Y_2) = \hat{e}(\sigma'_1, Y_2)^k$ is correct
3) If the signature of knowledge is valid, then the verifier returns 1, else 0

GOpen($GMSK, \mu, m, Reg[i]$):

Inputs: $GMSK, \mu, m, [i, \tau, \eta, \tau_2]$ where $\mu \leftarrow (\sigma'_1, \sigma'_2, c, s)$
Output: $i(userid), \pi$

1) Group Manager: Given $\mu = (\sigma'_1, \sigma'_2, c, s)$ and m for each element in Reg, using $Reg[i] = [i, \tau, \eta, \tau_2]$, check if $\hat{e}(\sigma'_2, g_2) \cdot \hat{e}(\sigma'_1, X_2)^{-1} = \hat{e}(\sigma'_1, \tau_2)$
2) Group Manager: outputs corresponding (i, τ, η) and SPK (Signature Proof of Knowledge) for τ_2