Distributed Broadcast Encryption for Confidential Interoperability across Private Blockchains *

Angelo De Caro[1], Kaoutar Elkhiyaoui[2], Sandeep Nishad[3], Sikhar Patranabis[4], and Venkatraman Ramakrishna[5]

[1,2]IBM Research Zürich
[3,4,5]IBM Research India

Abstract

Interoperation across distributed ledger technology (DLT) networks hinges upon the secure transmission of ledger state from one network to another. This is especially challenging for *private* networks whose ledger access is limited to enrolled members. Existing approaches rely on a *trusted centralized* proxy that receives encrypted ledger state of a network, decrypts it, and sends it to members of another network. Though effective, this approach goes against the founding principle of DLT, namely avoiding single points of failure (or single sources of trust).

In this paper, we leverage *fully-distributed* broadcast encryption (FDBE in short) to build a fully decentralized protocol for *confidential information-sharing* across private networks. Compared to traditional broadcast encryption (BE), FDBE is characterized by *distributed setup and key generation*, where mutually distrusting parties agree on a BE's public key without a trusted setup, and *securely* derive their decryption keys. Given any FDBE, two private networks can securely share information as follows: a sender in one network uses the other network's FDBE public key to encrypt a message for its members; and the resulting construction is secure in the simplified universal composability framework.

To further demonstrate the practicality of our approach, we present the first instantiation of an FDBE that enjoys constant-sized decryption keys and ciphertexts, and evaluate the resulting performances through a reference implementation that considers two private Hyperledger Fabric networks within the Hyperledger Cacti interoperation framework.

*An extended abstract of this paper will appear in the proceedings of NDSS 2026 (published by the Internet Society). This is the full version of the paper.

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 5
- 1.2 Technical Overview of Our FDBE Scheme . . . 7
- 1.3 Additional Related Work . . . 8

2 Preliminaries **9**
- 2.1 Bilinear Pairings . . . 9
- 2.2 Additional Cryptographic Background . . . 12

3 Modeling Confidential Cross-Network Communication **14**
- 3.1 Notations and Background Assumptions . . . 14
- 3.2 The Ideal Functionality $\mathcal{F}_{\mathsf{CN}}$ . . . 15

4 Realizing $\mathcal{F}_{\mathsf{CN}}$ using Fully Distributed Broadcast Encryption (FDBE) **17**
- 4.1 Fully Distributed Broadcast Encryption (FDBE) . . . 17
- 4.2 Protocol Realizing $\mathcal{F}_{\mathsf{CN}}$ . . . 19
- 4.3 Bilinear Pairing-based Construction of FDBE . . . 22
- 4.4 Analysis of Correctness . . . 26
- 4.5 Analysis of Key Extractability . . . 27
- 4.6 Analysis of Confidentiality . . . 28
- 4.7 Additional Discussion . . . 31
- 4.8 Supporting Dynamic Addition of Parties . . . 32

5 Benchmarks and Application **33**
- 5.1 Benchmarking Results for FDBE . . . 33
- 5.2 Benchmarking Results for CN-FDBE . . . 36

6 Conclusion **42**

A Application and System-Building: Additional Details **49**
- A.1 Cacti Weaver Chaincode Augmentation for CN-FDBE . . . 49
- A.2 CN-FDBE Protocol . . . 51

1 Introduction

Blockchain and distributed ledger technology (DLT) networks were originally designed for public access and open participation, as evidenced by the popular Bitcoin [Nak08] and Ethereum [W+14], and more recently, by networks like Algorand [CM19] and Solana [Yak18]. Though ideal for global cryptocurrency transactions, this model cannot support commerce involving digital assets where parties need more privacy, performance, and auditability than what public DLTs offer. Therefore, about a decade ago, interest in *private* or *permissioned* DLTs arose within enterprises and governmental institutions, which started to adapt and use the decentralized consensus-based shared ledger as a shared system-of-record for transactions among members of private groups (or business consortiums) [ABB+18, ABC+23]. Private DLTs, in combinations with public ones, have enabled applications like supply chains [gsb, tra, ibm], trade finance [mar, wet], central bank digital currency (CBDC) [sin, bdf21], securities trading [dtca, bdf21], and regulatory compliance [BIL+18] to be built at scale. In all of these scenarios, mutually untrusting parties (like exporters, importers, and banks) could not rely on common trusted central authorities to manage ledgers, yet needed a trustworthy decentralized transaction processing method with audit trails for dispute resolutions.

DLT networks with different trust and governance models have proliferated, both of the public and private forms, the latter typically serving business consortiums. This in turn has driven research in interoperability, which is a necessity if these networks are to fulfil their potential and serve useful functions. Interoperability allows networks with interdependent business processes to link with each other, break siloes, and manage digital assets jointly [bdf21, hsb22, hql22]. Technically, interoperation enables transactions to span multiple networks, primarily to (i) share state and drive operations [ABG+19], (ii) exchange assets atomically [htl20, NRVN22] (e.g., delivery-vs-payment (DvP) [bdf21, hsb22, hql22]), and (iii) transfer assets securely [HHSR23, dtcb]. All these patterns at their core rely on foolproof cross-network communication [ZABZ+19], or the ability to convey information held in one distributed ledger to another. A unique feature of cross-blockchain/DLT network communication is that the senders and receivers of information are decentralized networks, or mutually mistrustful groups of entities *without shared trusted* proxies or spokespersons [MKC22].

Communication solutions invariably involve *intermediary* components like bridges [BSF+24], settlement chains [Woo16, KB16], relay nodes [ABG+19], and even global messaging systems like SWIFT [dtcb]. Some of these components are external to the intercommunicating networks whereas others lie at least partly within those networks (i.e., having partial access to the network's resources). Though designed for both public and private DLT networks, such intermediaries pose unique challenges and threats to the latter that are not faced in the former. This is because a private network is not simply a constrained form of a public network but is a distinct class of distributed systems. Private networks' ledger states and transaction logs, unlike those of public networks, are not visible to nor verifiable from the external world by default. But external visibility and verifiability are (by definition) *key requirements* for cross-network communication, without which transactions across DLT networks cannot be fulfilled.

The intermediaries listed earlier fulfil these requirements but introduce other threats, namely the ability to tamper with data integrity (this is relevant to both public and private networks) and to exfiltrate a network's ledger data (this is relevant only to private networks). Let us examine the latter threat closely. By design, a private DLT network's data is kept confidential within its group of members. Such networks have internal consensus protocols to overcome faulty or malicious members, just like public networks do. This threat model covers network nodes tampering with ledger and transaction integrity but does not cover private ledger state leakage to entities outside

the group. New threat vectors are introduced when private networks must communicate private ledger data back-and-forth via intermediary components, which cannot be trusted to maintain the privacy of communicated data, unlike the network's nodes (group members). Prior interoperability research has focused on ensuring end-to-end communication integrity [1] as this impacts both public and private networks. Ensuring end-to-end communication confidentiality, which only impacts private networks' abilities to engage in cross-network transactions, has received less attention.

In a typical end-to-end cross-network communication instance (see Figure 1), an intermediary component extracts information from a *source network's* nodes, typically via smart contracts. This information is conveyed (often via other intermediary components) to a subset of the nodes of a *destination network*, which process this information and update their ledger (again, via smart contracts). Because these are DLT networks, the smart contracts may only run deterministic procedures, preventing them (or the nodes that host them) from being active participants or initiators of operations or from establishing communications with external components. Instead, they are only allowed to process transactions and queries submitted by clients (or Layer-2 entities) with network-issued credentials. It is possible then for these clients and other communication intermediaries (e.g., bridges, relays) to tamper with the information (motivating the need for end-to-end integrity) or exfiltrate the information to unauthorized external entities (motivating the need for end-to-end confidentiality).

Research on blockchain and DLT interoperability has focused primarily on cross-network communication and settlement mechanisms, and generating proofs of ledger state that can be independently validated by parties external to the network [twp, BCD+14, dog, TSB19, Mil12, KLS16, KMZ20]. This problem has a shared technical challenge with the scalability problem in public blockchains, which has been solved by a variety of techniques typically involving *sidechains* that handle a smaller portion of the workload separate from the main chain [PD15, PB17, NPS20, zkp21, TSH22]. To summarize, ensuring the integrity of communication across DLT networks/chains (whether independent networks or main-chain/sidechain) has been well-researched in recent years. However, the challenge of providing confidentiality in addition to integrity has not been considered. The only prior work that tackles the confidentiality problem to some extent is the data sharing protocol provided by Hyperledger Cacti [Cac25, Cacd], but as mentioned earlier, this realizes a weaker security model involving a centralized trust assumption on a network client/proxy (more on this later).

We tackle the open question of enabling confidential yet decentralized communication across (private) blockchain/DLT networks *without relying on trusted intermediaries*. Our goal is to enable the following: a group of nodes in the source network should be able to agree on some content and then send it across in a confidential manner such that the content can only be accessed (and subsequently, validated and agreed upon) by a designated group of receiver nodes in the destination network. One can further separate the goals of ensuring agreement and content validity from that of confidentiality. The first two challenges, which impact end-to-end integrity, are addressed through existing decentralized solutions such as distributed consensus [XWJ23] and threshold signatures [Sho00]. The core challenge that we address is *confidential cross-network communication* (with integrity assurance), where the sender (which may be a single entity or a group of entities) can efficiently and confidentially share private content with a specific group of receivers.

We note that such a mechanism is likely to be of broader interest to any setting involving groups sending or receiving private information, including (but not restricted to) private DLT

[1] By "end-to-end integrity", we mean that information possessed by the nodes in one network must reach relevant nodes of another network without intermediaries being capable of tampering with the information.

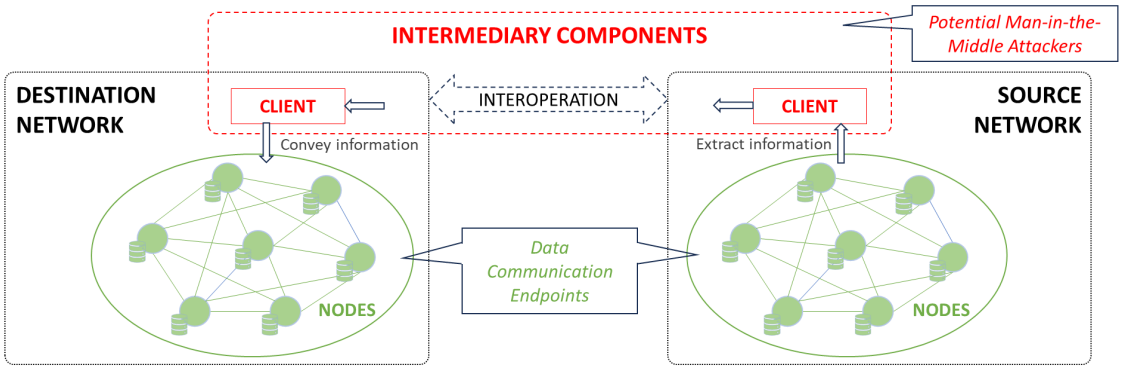


Figure 1: Cross-Network Communication Model for Private DLTs

interoperability. Sensors and end user devices in IoT pipelines can use it to send information confidentially to selected stream processors and data warehouses via untrusted message brokers. Users and groups in a social networking platform like WhatsApp [Pla] can use it to send messages confidentially to targeted members of other groups without creating a new group for every possible subset of users (which will introduce cognitive overhead and scalability challenges).

Drawbacks of Existing Approaches. Known techniques for confidential cross-network communication either require the sender to encrypt the message separately for each recipient in the destination network, or rely on a dedicated (semi-trusted) network proxy to disseminate the message to the intended set of recipients. The first approach does not have any practical instances to the best of our knowledge, since typical private DLT networks disallow direct exchange of cross-network messages between individual parties. This approach is also inefficient with respect to bandwidth consumption, as the sender needs to send multiple ciphertexts, and must incorporate all of the destination public keys (possibly wrapped in certificates) in the payload.

For the second approach, the closest practical example that we encountered [Cac25, Cacd] uses a semi-trusted network client as a proxy that decrypts the confidential data on behalf of the recipient group and then disseminates it to the intended recipients. This approach inherently requires a (centralized) trust assumption on the proxy for data confidentiality and integrity, which goes against the founding principle of DLT, namely avoiding single points of failure. In particular, a malicious proxy with the ability to decrypt data from another ledger could exfiltrate that data to unauthorized third parties instead of just submitting the data to its network's peers (as intended). This motivates us to ask: *can we efficiently realize confidential yet fully decentralized cross-network communication?*

1.1 Our Contributions

We answer the above question in the affirmative, and introduce a novel, fully decentralized, and provably secure protocol for confidential and authenticated cross-network communication, with low bandwidth requirements and key management overheads. This enables the first (to the best of our knowledge) framework for confidential communication across private DLT networks that does not rely on trusted intermediaries. We summarize our key technical contributions below.

Modeling Confidential Cross-Network Communication. We formally model confidential and decentralized communication across private DLT networks as an ideal functionality in the

Scheme	Setup	$\|\mathsf{pp}\|$	$\|\mathsf{sk}\|$	$\|\mathsf{ct}\|$
[WQZD10]	Decentralized	$O(n^2)$	$O(n)$	$O(1)$
[KMW23]-1	Trusted	$O(n)$	$O(1)$	$O(1)$
[KMW23]-2	Trusted	$O(n^2)$	$O(1)$	$O(1)$
[CGPW25, CGPP24, GKPW24]	Trusted	$O(n)$	$O(1)$	$O(1)$
FDBE (this work)	Decentralized	$O(n)$	$O(1)$	$O(1)$

Table 1: Comparison of bilinear pairing-based BE schemes with distributed key generation. Here n denotes the total number of parties, $|\mathsf{pp}|$ denotes the size of the public parameters (including the public key), $|\mathsf{sk}|$ denotes the size of the decryption key for each party, and $|\mathsf{ct}|$ denotes the size of the ciphertext. [CGPW25] and [CGPP24] essentially use the same underlying DBE scheme proposed in [GKPW24], so we group them together.

simplified universal composability (sUC) framework from [CCL15]. As noted in [CCL15], the sUC framework is a practically meaningful, easy-to-use alternative to the traditional UC framework from [Can01]. For simplicity of exposition, our functionality uses a simplified abstraction of a private DLT networks (and the corresponding private ledgers), and focuses on capturing the core data confidentiality requirements. The detailed description appears in Section 3.

Realization from Fully Distributed Broadcast Encryption. We present a protocol that securely emulates the above functionality while relying (in a black-box manner) on a *fully decentralized* version of Broadcast Encryption (BE) [FN94,DF03,BGW05,BW06,BZ14,GKW18,KMW23, GKPW24, CGPP24, CGPW25] (see Section 1.3.2 for a detailed treatment of related works on BE and its distributed variants). We call this primitive fully distributed BE (or FDBE in short). We formally define FDBE in Section 4.1. We then describe our FDBE-based protocol and prove its security in the sUC framework in Section 4.2.

Concrete Instantiation of FDBE. We present the first FDBE scheme with constant-sized keys and ciphertexts that supports fully decentralized setup and distributed key generation, achieves comparable encryption and decryption efficiency to the most efficient, trusted-setup based BE schemes (such as [BGW05,BW06,GKW18]), relies on well-studied cryptographic hardness assumptions over bilinear groups, and is suitable for deployment in private blockchain/DLT networks (we expand more on this in the discussion below). Plugging this into our generic construction yields an instantiation of confidential and decentralized cross-network communication from the same assumption with low bandwidth requirements and small key management overheads. In particular, the encryption and decryption overheads of our solution depend only on the broadcast-group size (not the overall network-size), the ciphertext sizes are constant, and the overheads of computing/storing/updating decryption-keys are also constant. See Section 4.3 for the detailed construction of our FDBE scheme.

In Table 1, we compare our FDBE scheme to recently proposed pairing-based BE constructions with distributed key generation [KMW23, GKPW24, CGPP24, CGPW25] (we note that [CGPW25] and [CGPP24] essentially use the same underlying DBE scheme proposed in [GKPW24]). While these schemes match our FDBE scheme in terms of efficiency, all of them require a (one-time) trusted setup to generate a common reference string. To the best of our knowledge, the only other FDBE scheme in the literature with a fully distributed setup was proposed in [WQZD10]. However,

[WQZD10] has $O(n^2)$-sized public parameters and $O(n)$-sized keys (n being the total number of parties). The corresponding overheads for our FDBE scheme are $O(n)$ and $O(1)$, respectively, resulting in significantly greater practical efficiency.

The main technical challenge we overcome in designing our FDBE scheme is distributing the setup and key generation procedures while preserving *collusion-resistance*, which is the core security property of broadcast encryption. Informally, collusion resistance in FDBE requires that party-i alone is able to compute her secret decryption key given the public parameters and her own share of the master secret key. It turns out that naïvely decentralizing the setup procedure in existing BE schemes such as [BGW05] using techniques such as secure multi-party computation (MPC) does not natively maintain semantic security against collusions. We address this in our FDBE scheme by carefully augmenting the public parameters in the original scheme of [BGW05] to include some additional terms, such that the setup can be decentralized while (provably) ensuring that no party other than party-i can compute her decryption key.

A natural question to ask is if one could obtain FDBE schemes by distributing the trusted setup procedure in [KMW23, GKPW24, CGPP24, CGPW25] using MPC. While this is theoretically feasible, deploying this in private blockchain/DLT networks (which is our target use-case in this paper) incurs major practical barriers. In particular, generic MPC protocols often rely crucially on point-to-point communication channels between each pair of parties. This requirement is fundamentally incompatible with the native infrastructure of existing private blockchain/DLT networks. Our FDBE scheme avoids this issue by decentralizing the setup in a round-robin manner, where each (sequential) round only requires the involvement of a single party, and the corresponding output is written to the private blockchain/distributed ledger (thereby avoiding the need for point-to-point communication channels between parties). The sequential nature of our distributed setup protocol is, in fact, ideally compatible with the inherently sequential nature of writing to a private blockchain/distributed ledger.

Additionally, generic MPC protocols only achieve abort security when a majority of the parties are maliciously corrupt [Cle86]. Abort security does not guarantee that a cheating party would be identified; even if a cheating party is identified and discarded, the protocol would require restarting from scratch. Our FDBE scheme achieves a stronger notion of malicious security where *each* cheating party is identified, and setup can simply proceed by discarding the corresponding round, without re-starting from scratch.

Implementation and Benchmarking. We implement and benchmark our proposed FDBE scheme and concretely compare, in Section 5.1, its performance against the pairing-based BE schemes from Table 1. We further demonstrate the practicality of the FDBE-based instantiation, which we call henceforth Cross-Network-FDBE (abbreviated as CN-FDBE), through a reference implementation that enables confidential interoperation across two private Fabric networks within the Hyperledger Cacti interoperation framework. We show that overheads introduced by FDBE-based communication are negligible or tolerable for networks of practically realistic sizes. See Section 5.2 for details.

1.2 Technical Overview of Our FDBE Scheme

In this section, we present a brief overview of our construction of FDBE, which is the technical core of our proposed solution for confidential data sharing across permissioned ledgers.

Background. We briefly introduce some notations for ease of exposition. Let $\mathbb{G}$ be a group of prime order p that admits a non-degenerate bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and let P and Q

be two uniformly-random generators for $\mathbb{G}$. We use upper-case letters to refer to elements in $\mathbb{G}$, whereas lower-case letters are used to refer to elements in $\mathbb{F}_p$. The starting point of our FDBE solution is the bilinear pairing-based BE scheme from [BGW05], which allows a sender to compute a single (constant-sized) ciphertext for a set of recipients, such that *only* the designated recipients can decrypt the ciphertext using their *own private* (constant-sized) decryption keys (i.e., without any shared decryption key). However, this scheme crucially relies on a trusted, centralized setup mechanism that generates a public key pk and the recipients' decryption keys $\mathsf{dk}_1, \ldots, \mathsf{dk}_n$ (where n is the maximum number of recipient slots). More specifically:

$$\mathsf{pk} = (P, P^x, ..., P^{x^n}, P^{x^{n+2}}, ..., P^{x^{2n}}, P^y) \; ; \; \mathsf{dk}_i = P^{yx^i}$$

where x and y are secret keys/trapdoors known only to the centralized authority.

Distributing Setup and Key Generation. In order to meet the requirements of FDBE (as formalized in Section 4.1), we must (informally speaking) devise a mechanism that distributes the generation of pk and dk_i, thereby removing the requirement of a central authority that knows the secret trapdoor. To begin with, we observe that $(P, P^x, ..., P^{x^n}, P^{x^{n+2}}, ..., P^{x^{2n}})$ and (P^y, P^{yx^i}) can be distributively produced using known techniques for updatable structured reference strings (SRS) and distributed key generation, respectively, from the threshold cryptography literature. However, in the context of FDBE, a unique challenge arises when factoring the requirement that *only the rightful holder of* dk_i *should be able to compute it.* In particular, a naïve adaptation of techniques for distributed generation of SRS and keys does not meet this requirement.

Variant of [BGW05]. To address this challenge, we first introduce a new variant of the BE scheme from [BGW05]. In this variant, we define the public key pk as $(P, P^x, ..., P^{x^n}, P^{x^{n+2}}, ..., P^{x^{2n}}, Q)$ and the decryption keys dk_i as Q^{x^i}, where the discrete logarithm of Q relative to P is unknown to all system participants. We then define a structured reference string that extends the public key of the BE scheme with auxiliary information:

$$\mathsf{aux} = (Q^{x/x_1}, Q^{x^2/x_2^2}..., Q^{x^n/x_n^n}), \quad x = \prod_{i=1}^{n} x_i$$

Notice that if the value x_i is only known to the participant assigned decryption key dk_i, then this guarantees that no one but that participant can compute it.

Distributing Setup and Key Generation for New Variant. At this point, we leverage techniques from the literature on updatable SRS to compute pk and aux, which combined make the public parameters of the scheme. The key technical challenge that we address here is ensuring that *no party but the holder of decryption key* dk_i *can compute it.* At a high level, we achieve this by ensuring that only the holder of dk_i knows x_i, but no one else, and that knowledge of x_i is essential to computing dk_i. We refer to Section 4.3 for the details.

1.3 Additional Related Work

In this section, we present a detailed treatment of additional related work.

1.3.1 UC Models for Blockchain/DLT Solutions

Several works [GKL24,BGK+18,GRR+21,CKS24] have modeled various aspects of blockchain/DLT-based solutions in the universal composability (UC) framework and its variants [Can01, CCL15,

CKKR19]. In this paper, we use the UC formalization of DLTs in [BGK+18] as a black-box and build upon it to introduce a (simplified) UC formalization of cross-(private)network communication protocol, where each network is equipped with a ledger, such that the ledger state is maintained by a group of nodes.

1.3.2 Broadcast Encryption

In this subsection, we discuss relevant related works on broadcast encryption (BE) and its distributed variants.

Broadcast Encryption. BE enables a sender to encrypt and broadcast a message to a group of recipients efficiently. That is, the size of the broadcasted ciphertext is sub-linear in the number of intended recipients. Furthermore, a secure broadcast encryption guarantees that even if everyone outside the sender and the recipients collude, they cannot learn the content of the broadcasted message. Following the earliest works on symmetric-key BE [FN94, HS02], numerous works have studied public-key BE [DF03, BGW05, BW06, GW09, GKW18] with a focus on optimizing the size of ciphertexts and decryption keys, as well as weakening the security assumptions. A long line of works have also studied BE schemes based on ciphertext-policy attribute-based encryption, resulting in constructions with optimal parameters, including constant-sized public parameters [BSW07, CGW15, AWY20]. All of these schemes inherently require *centralized trust assumptions*, including *trusted setup* and *key escrow*, where a centrally trusted entity holding the master secret key issues decryption keys to individual recipients.

Distributed BE. Several works [BZ14, FWW23, KMW23, GKPW24, CGPP24, CGPW25] have proposed removing key escrow in BE via distributed key generation without a centrally trusted authority. Naïve approaches based on public-key encryption and registration-based encryption incur sub-optimal ciphertext sizes that grow with the size of the target broadcast set. Some other proposals [BZ14, FWW23] rely on heavy cryptographic machinery such as indistinguishability obfuscation and witness encryption, that are not amenable to practical instantiations. As discussed in Section 1.1 and summarized in Table 1, a recent line of works have proposed elegant, pairing-based BE constructions with distributed key generation [KMW23, GKPW24, CGPP24, CGPW25], but they require trusted and centralized setup. On the other hand, our FDBE scheme supports both decentralized setup and distributed key generation, while being significantly more efficient than prior FDBE schemes, such as [WQZD10].

2 Preliminaries

In this section, we present preliminary background material.

2.1 Bilinear Pairings

In this section, we present additional background material on bilinear pairings, which are used crucially in our FDBE construction in Section 4.3. For simplicity of exposition, we use *symmetric* bilinear pairings to present the construction. Hence, we discuss symmetric bilinear pairings first. We subsequently discuss asymmetric bilinear pairings, which allow more efficient instantiations and are used in our prototype implementations and benchmarks.

Symmetric Bilinear Pairing. Informally speaking, in a symmetric bilinear pairing, the pairing map is a bilinear function that takes as inputs two elements from the same group $\mathbb{G}$ and outputs an element in the target group $\mathbb{G}_T$. The pairing map should additionally satisfy certain non-degeneracy and computational efficiency properties to be useful in cryptographic applications. We present a more detailed exposition below.

Definition 1 (Symmetric Bilinear Pairing)**.** *Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic groups of known prime order p (where p is typically a $O(\kappa)$-bit prime for security parameter κ). A* symmetric bilinear pairing *is a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ satisfying the following properties:*

- ***Bilinearity:*** *For all $P, Q \in \mathbb{G}$ and all $a, b \in \mathbb{F}_p$, the pairing satisfies*

$$e(P^a, Q^b) = e(P, Q)^{ab}.$$

  Equivalently, the pairing map e is linear in each argument, i.e., for all $P, P', Q, Q' \in \mathbb{G}$, the pairing satisfies

$$e(P \cdot P', Q) = e(P, Q) \cdot e(P', Q)$$
$$e(P, Q \cdot Q') = e(P, Q) \cdot e(P, Q')$$

- ***Non-degeneracy:*** *If $P \in \mathbb{G}$ is a non-identity element of $\mathbb{G}$, then*

$$\exists Q \in \mathbb{G} : e(P, Q) \neq 1.$$

  That is, the pairing is not the trivial map that always outputs the identity in $\mathbb{G}_T$.

- ***Efficient computability:*** *There exists an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in \mathbb{G}$.*

Cryptographic Realizations. In cryptographic applications, the group $\mathbb{G}$ and $\mathbb{G}_T$ are typically realized using elliptic curves. Specifically:

- The group $\mathbb{G}$ is typically realized as an additive subgroup of the group of elliptic curve points $E(\mathbb{F}_q)$ of order p.
- The group $\mathbb{G}_T$ is typically realized as an order-p multiplicative subgroup of a finite extension field $\mathbb{F}_{q^k}^{\times}$, where k is the *embedding degree*.

A symmetric pairing can be instantiated using functions such as the Weil pairing [Wei38, Mil04] or the Tate pairing [Tat74, Sil09]. Symmetric pairings are also sometimes referred to as Type-1 pairings.

Asymmetric Bilinear Pairings. Informally speaking, in an asymmetric bilinear pairing, the pairing map takes as inputs elements from two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ that are not necessary identical (unlike in the symmetric case, where the inputs to the pairing map are from the same group). We present a more detailed exposition below.

Definition 2 (Asymmetric Bilinear Pairings)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of known prime order p (where p is typically a $O(\kappa)$-bit prime for security parameter κ). An* asymmetric bilinear pairing *is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying the following properties:*

Curve Family	Example	Security Level	Embedding Degree k
MNT [MNT01]	MNT6	$\approx$ 80-bit	$k = 6$
BN [BN05]	BN254	$\approx$ 110-bit	$k = 12$
BLS [BKLS02]	BLS12-381 [Bow17]	$\approx$ 128-bit	$k = 12$
	BLS24	$\approx$ 192-bit	$k = 24$
Brezing-Weng [BW05]	KSS18 [KSS08]	$\approx$ 192-bit	$k = 18$

Table 2: A non-exhaustive list of pairing-friendly elliptic curves and their embedding degrees. The MNT [MNT01] family of curves is now widely deprecated due to insufficient security.

- ***Bilinearity:*** *For all $(P, Q) \in \mathbb{G}_1 \times \mathbb{G}_2$ and all $a, b \in \mathbb{F}_p$, the pairing satisfies*

$$e(P^a, Q^b) = e(P, Q)^{ab}.$$

  Equivalently, the pairing map e is linear in each argument, i.e., for all $P, P' \in \mathbb{G}_1$ and all $Q, Q' \in \mathbb{G}_2$, the pairing satisfies

$$e(P \cdot P', Q) = e(P, Q) \cdot e(P', Q)$$

$$e(P, Q \cdot Q') = e(P, Q) \cdot e(P, Q')$$

- ***Non-degeneracy:*** *If $P \in \mathbb{G}_1$ is a non-identity element of $\mathbb{G}_1$, then*

$$\exists Q \in \mathbb{G}_2 : e(P, Q) \neq 1.$$

  Similarly, if $Q \in \mathbb{G}_2$ is a non-identity element of $\mathbb{G}_2$, then

$$\exists P \in \mathbb{G}_1 : e(P, Q) \neq 1.$$

  In other words, the pairing is not the trivial map that always outputs the identity in $\mathbb{G}_T$.

- ***Efficient computability:*** *There exists an efficient algorithm to compute $e(P, Q)$ for all $(P, Q) \in \mathbb{G}_1 \times \mathbb{G}_2$.*

Categories. Asymmetric pairings are commonly categorized into Type 2 and Type 3:

- **Type-2:** There exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$, but not conversely.
- **Type-3:** No efficient homomorphisms exist between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Cryptographic Realizations. Type-3 pairings are typically preferred in practice due to better security and efficiency. In many cryptographic constructions, the groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of a Type-3 pairing are realized from elliptic curves:

- $\mathbb{G}_1$ is realized as an order-p additive subgroup of $E(\mathbb{F}_q)$ (same as in symmetric bilinear pairings).
- The second source group $\mathbb{G}_2$ is often realized as an order-p additive subgroup of $E(\mathbb{F}_{q^k})$ (k being the embedding degree).

- $\mathbb{G}_T$ is often realized as an order-p multiplicative subgroup of $\mathbb{F}_{q^k}^{\times}$.

In practice, Type-3 pairings such that (optimal) Ate pairing [HSV06, Ver10] are implemented using elliptic curves with a small embedding degree k and a large prime-order subgroup. Common examples include the Barreto–Lynn–Scott (BLS) family of elliptic curves [BKLS02] and the Barreto–Naehrig (BN) family of elliptic curves [BN05]. We summarize a non-exhaustive list of elliptic curves popularly used in pairing-based cryptographic applications along with their security levels in Table 2. See [AFG24] for a more detailed survey.

Remark. We remark here that symmetric bilinear pairings are often used to describe cryptographic applications for simplicity of exposition (see [BF01, BF03, BLS04, BGW05, BW06] for a non-exhaustive list of examples). We use the same approach in Section 4.3 to describe the construction of FDBE. However, in practice, asymmetric bilinear pairings allow for more efficient instantiations, and are more widely used for actually implementing pairing-based cryptographic systems. Accordingly, our implementation of FDBE also uses asymmetric bilinear pairings.

2.2 Additional Cryptographic Background

In this section, we present some preliminary background material on cryptographic primitives used in our construction.

2.2.1 Symmetric-Key Encryption

We recall the formal definition of symmetric-key encryption (SKE) below.

Definition 3 (Symmetric-Key Encryption)**.** *A symmetric-key encryption (SKE) scheme with κ-bit keys (κ being the security parameter) is a tuple of polynomial-time algorithms $(\mathcal{E}, \mathcal{D})$ described as follows:*

- $C \leftarrow \mathcal{E}(K, m)$*: a randomized algorithm that takes as input $K \in \{0,1\}^\kappa$ and a message $m \in \{0,1\}^{O(\kappa)}$, and generates a ciphertext C.*
- $m \leftarrow \mathcal{D}(K, C)$*: a deterministic algorithm that, on input the secret key K and a ciphertext C, outputs a message m.*

We require an SKE scheme to satisfy correctness and IND-CPA security as described below.

Correctness. *We say that a symmetric-key encryption scheme $(\mathcal{E}, \mathcal{D})$ is correct if for any security parameter $\kappa \in \mathbb{N}$, any $K \in \{0,1\}^\kappa$, and any $m \in \{0,1\}^{O(\kappa)}$, we have,*

$$\mathcal{D}(K, \mathcal{E}(K, m)) = m$$

IND-CPA Security. *We say that a symmetric-key encryption scheme $(\mathcal{E}, \mathcal{D})$ is IND-CPA secure if for any security parameter $\kappa \in \mathbb{N}$, any $K \leftarrow \{0,1\}^\kappa$, any $m_0, m_1 \in \{0,1\}^{O(\kappa)}$, and any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, letting γ_b for $b \in \{0,1\}$ denote the following probability*

$$\gamma_b = \Pr\left[\mathcal{A}^{\mathcal{E}(K,\cdot)}(1^\kappa, \mathcal{E}(K, m_b)) \to 0\right]$$

we have

$$|\gamma_0 - \gamma_1| \leq \mathsf{negl}(\kappa)$$

2.2.2 Non-Interactive Zero-Knowledge (NIZK) Arguments of Knowledge

In this subsection, we recall the formal definition of a non-interactive zero-knowledge (NIZK) arguments of knowledge.

Definition 4 (Ternary Relation)**.** *A ternary relation $\mathcal{R}$ is defined by a triple $(\mathsf{pp}, \mathbb{x}, \mathbb{w})$ where pp is the public parameters, $\mathbb{x}$ the instance, and $\mathbb{w}$ the witness. If triple $(\mathsf{pp}, \mathbb{x}, \mathbb{w})$ satisfies $\mathcal{R}$, then we write $\mathcal{R}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) = 1$. Else $\mathcal{R}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) = 0$. We refer to $\mathcal{L}_{\mathcal{R}} = \{(\mathsf{pp}, \mathbb{x}) : \exists \mathbb{w} \text{ s.t. } \mathcal{R}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) = 1\}$ as the language of relation $\mathcal{R}$.*

Definition 5 (Interactive Argument of Knowledge)**.** *Let $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ be the following three algorithms.*

- **Generator** $\mathcal{G}$ *takes as input of security parameter 1^κ and a description of relation $\mathcal{R}$ and returns public parameters pp.*
- **Prover** $\mathcal{P}$ *takes as input pp, $\mathbb{x}$ and $\mathbb{w}$, whereas* **verifier** $\mathcal{V}$ *takes as input pp and $\mathbb{x}$. $\mathcal{P}$ and $\mathcal{V}$ are interactive algorithms, whose joint interaction results in a transcript $\mathsf{tr} \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle$.*
  The interaction between $\mathcal{P}$ and $\mathcal{V}$ concludes by having $\mathcal{V}$ output a bit $b = \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle$. $b = 1$ indicates that tr is accepted by $\mathcal{V}$; otherwise, tr is rejected.

The triple $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ defines an interactive argument of knowledge if it satisfies the following properties.

Completeness. *$(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is complete if for all security parameters $\kappa \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$:*

$$\Pr\left[\begin{array}{c|c} \mathcal{R}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) = 0 & \mathsf{pp} \leftarrow \mathcal{G}(1^\kappa, \mathcal{R}) \\ \vee & (\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle = 1 & \end{array}\right] = 1 \ .$$

Knowledge Soundness. *$(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is knowledge-sound if for all security parameters $\kappa \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, there exists an extractor $\mathcal{X}$ such that:*

$$\Pr\left[\begin{array}{c|c} \mathcal{R}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) = 0 & \mathsf{pp} \leftarrow \mathcal{G}(1^\kappa, \mathcal{R}) \\ \wedge & (\mathsf{st}, \mathbb{x}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \langle \mathcal{A}(\mathsf{st}, \mathbb{x}), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle = 1 & \mathbb{w} \leftarrow \mathcal{X}^{\mathcal{A}(\mathsf{st}, \mathbb{x})}(\mathsf{pp}) \end{array}\right] \leq \mathsf{negl}(\kappa) \ .$$

Definition 6 (Public-Coin Interactive Arguments of Knowledge)**.** *An interactive argument of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is public-coin if all messages that $\mathcal{V}$ sends to $\mathcal{P}$ are generated uniformly at random. In other words, $\mathcal{V}$'s messages to $\mathcal{P}$ (called also challenges) correspond to $\mathcal{V}$'s randomness.*

A public-coin interactive argument of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is zero-knowledge if: $\mathsf{tr} \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle$ leaks zero information about the witness $\mathbb{w}$. More formally:

Definition 7 (Zero-knowledge)**.** *$(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is zero-knowledge if for all security parameters $\kappa \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that the following holds:*

$$\Pr\left[\begin{array}{c|c} (\mathsf{pp}, \mathbb{x}) \in \mathcal{L}_{\mathcal{R}} & \mathsf{pp} \leftarrow \mathcal{G}(1^\kappa, \mathcal{R}) \\ \wedge & (\mathbb{x}, \mathbb{w}, \mathsf{chal}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathcal{A}(\mathsf{tr}) = 1 & \mathsf{tr} \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{pp}, \mathbb{x}; \mathsf{chal}) \rangle \end{array}\right]$$

$$\approx \Pr\left[\begin{array}{c|c} (\mathsf{pp}, \mathbb{x}) \in \mathcal{L}_{\mathcal{R}} & \mathsf{pp} \leftarrow \mathcal{G}(1^{\kappa}, \mathcal{R}) \\ \wedge & (\mathbb{x}, \mathsf{chal}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathcal{A}(\mathsf{tr}) = 1 & \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{pp}, \mathbb{x}, \mathsf{chal}) \end{array}\right] .$$

where chal *is the public-coin randomness of* $\mathcal{V}$.

Non-interactive Zero-knowledge (NIZK) Arguments of Knowledge. In the random oracle model (ROM), public-coin interactive (zero- knowledge) arguments of knowledge can be made non-interactive using the Fiat-Shamir heuristic [FS86]. In particular, in the case of Sigma protocols, the resulting arguments are defined by triple $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, such that: $\mathcal{G}(1^{\kappa}, \mathcal{R})$ outputs the public parameters pp that also contain a description of a hash function. $\mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w})$ computes the challenge as the hash of the first message to be sent to $\mathcal{V}$, and returns the corresponding proof Π. Finally, $\mathcal{V}(\mathsf{pp}, \mathbb{x}, \Pi)$ returns a bit b, where $b = 1$ signifies that Π is valid.

3 Modeling Confidential Cross-Network Communication

In this section, we formally model cross-network confidential communication as an ideal functionality $\mathcal{F}_{\mathsf{CN}}$ in the *simplified universal composability* (SUC) framework of [CCL15]. We first introduce some notations and background assumptions, and then describe $\mathcal{F}_{\mathsf{CN}}$.

3.1 Notations and Background Assumptions

Notations. Let $\mathcal{N}_0$ and $\mathcal{N}_1$ be two private DLT networks operated by parties $\mathcal{P}_0 = \{\mathfrak{p}_1, ..., \mathfrak{p}_{n_0}\}$ and $\mathcal{P}_1 = \{\mathfrak{p}'_1, ..., \mathfrak{p}'_{n_1}\}$ respectively. Given that $\mathcal{N}_0$ and $\mathcal{N}_1$ are permissioned networks, each party in $\mathcal{P}_0$ and $\mathcal{P}_1$ is equipped with a long-term identity that enables her identification and the verification of her authorizations. We do not make any assumptions on how the identities and the authorizations are verified. However, it's safe to assume that these operations will leverage a public-key infrastructure. As DLT networks, $\mathcal{N}_0$ and $\mathcal{N}_1$ maintain each a state ledger: $\mathcal{L}_0$ for $\mathcal{N}_0$ and $\mathcal{L}_1$ for $\mathcal{N}_1$. The ledgers are key-value stores and their updates are governed by the smart contracts running on top of the underlying DLT network. We denote by $\mathcal{L}_b[\mathsf{K}]$ the value stored in $\mathcal{L}_b$ at key K.

Background Assumptions. We assume that the parties in $\mathcal{P}_0$ and $\mathcal{P}_1$ collectively guarantee the *safety* and *liveness* of $\mathcal{N}_0$ and $\mathcal{N}_1$. Namely, if f_0 and f_1 are the corruption thresholds beyond which the security of the consensus protocol underlying $\mathcal{N}_0$ and $\mathcal{N}_1$ cannot be assured, then we assume that only f_0 and f_1 parties can be corrupted within $\mathcal{P}_0$ and $\mathcal{P}_1$. For simplicity, we consider that the smart contract deployment and execution are out of scope. However, we stress that thanks to the transparency and verifiability of DLT networks, each party in $\mathcal{P}_0$ and $\mathcal{P}_1$ can check for herself whether a deployed smart contract matches the agreed-upon specifications and whether the ledger state updates correspond to correct smart contract executions.

Communication Bridges. A cross-network communication protocol aims at allowing a subset of parties in $\mathcal{P}_0$ to exchange data with another subset of parties in $\mathcal{P}_1$, and vice versa. We refer to this operation as "RemoteRead" in contrast with "Read". The latter corresponds to a party in $\mathcal{P}_0$ (or $\mathcal{P}_1$) locally reading the value of a key in $\mathcal{L}_0$ (or $\mathcal{L}_1$). We consider, in the following, that each network independently defines the policies controlling remote access to the state of the ledger. We also consider, for simplicity, that an *honest* RemoteRead operation can only be performed if the intersection $\mathcal{P}_0 \cap \mathcal{P}_1 \neq \emptyset$. We call the non-empty intersection a *communication bridge*, which will

RemoteRead On query $(\mathsf{RemoteRead}, \mathsf{rid}, \mathsf{K}, \mathcal{R}, \mathcal{L}_b)$ from a party $\mathfrak{p} \in \mathcal{P}_{\bar{b}}$, send message $(\mathsf{RemoteRead}, \mathsf{rid}, \mathfrak{p}, \mathsf{K}, \mathcal{R}, \mathcal{L}_b)$ to $\mathcal{S}$ and await the go-ahead. On receiving the go-ahead do:

1. **if** $(\mathcal{P}_0 \cap \mathcal{P}_1) \subset \mathcal{C}$ **then** await $\mathcal{S}$'s response.
   (a) **if** $\mathcal{S}$ sends OK **then** continue.
   (b) **else** abort.
2. **else** continue.
3. send message $(\mathsf{RemoteRead}, \mathsf{rid}, \mathfrak{p}, \mathsf{K}, \mathcal{R})$ to $\mathcal{P}_b$.

SendValue On message $(\mathsf{SendValue}, \mathsf{rid}, \mathsf{K}, \mathcal{R}, \mathcal{L}_b, \mathsf{v})$ from party $\mathfrak{p} \in \mathcal{P}_b$, send message $(\mathsf{SendValue}, \mathsf{rid}, \mathsf{K}, \mathcal{R}, \mathcal{L}_b, \mathfrak{p})$ to $\mathcal{S}$ and await the go-ahead. On receiving the go-ahead do:

1. **if** $\mathfrak{p} \in \mathcal{P}_b \setminus \mathcal{C}$ **then** send message $(\mathsf{CheckAuth}, \mathfrak{p}, \mathsf{K}, \mathcal{R})$ to $\mathcal{N}_b$ and await the response.
   (a) **if** $\mathcal{N}_b$ sends OK **then** continue.
   (b) **else** abort.
2. **else** continue.
3. **if** $(\mathcal{P}_0 \cap \mathcal{P}_1) \subset \mathcal{C}$ **then** await $\mathcal{S}$'s response:
   (a) **if** $\mathcal{S}$ sends OK **then** continue.
   (b) **else** abort.
4. send message $(\mathsf{SendValue}, \mathsf{rid}, \mathsf{K}, \mathcal{L}_b, \mathfrak{p}, \mathsf{v})$ to $\mathcal{R}$.
5. **if** $(\mathcal{R} \cap \mathcal{C}) \neq \emptyset$ **then** send message $(\mathsf{SendValue}, \mathsf{rid}, \mathsf{v})$ to $\mathcal{S}$.

Figure 2: Ideal functionality $\mathcal{F}_{\mathsf{CN}}$ for confidential cross-ledger communication between two permissioned networks $\mathcal{N}_0$ and $\mathcal{N}_1$, operated by party sets $\mathcal{P}_0$ and $\mathcal{P}_1$ that maintain state ledgers $\mathcal{L}_0$ and $\mathcal{L}_1$ respectively.

relay RemoteRead requests between $\mathcal{N}_0$ and $\mathcal{N}_1$. If all the parties in the communication bridge are corrupt, then the communication between $\mathcal{P}_0$ and $\mathcal{P}_1$ can be disrupted, and we say that the bridge is corrupt. Indeed, a corrupt bridge may refuse to deliver messages from $\mathcal{P}_0$ to $\mathcal{P}_1$ and vice versa.

3.2 The Ideal Functionality $\mathcal{F}_{\mathsf{CN}}$

We require a cross-network communication protocol to be *correct* and *confidential*. Correctness refers to the property that if the bridge $\mathcal{P}_0 \cap \mathcal{P}_1$ is not corrupt, then any subset of parties in $\mathcal{P}_0$ ($\mathcal{P}_1$) that are authorized to RemoteRead the value of a key in $\mathcal{L}_1$ ($\mathcal{L}_0$) will always receive that value upon request. Confidentiality, on the other hand, refers to the property that honest parties in $\mathcal{P}_0$ and $\mathcal{P}_1$ will not inadvertently violate RemoteRead policies: i.e., an honest party in $\mathcal{P}_0$ ($\mathcal{P}_1$) will only share data with parties in $\mathcal{P}_1$ ($\mathcal{P}_0$) that satisfy the RemoteRead policies. Finally, cross-network communication should be *authenticated*: the party receiving data in one network should be able to authenticate its source in the other network; this is, usually, addressed using signatures.

Modeling in Simplified UC. To formally capture these three properties, we turn to the SUC framework [CCL15], which leverages the real/ideal world paradigm. In the ideal world, we find an ideal functionality $\mathcal{F}_{\mathsf{CN}}$ that satisfies the desired security properties by construction, and a simulator $\mathcal{S}$ that corrupts parties in both $\mathcal{P}_0$ and $\mathcal{P}_1$. $\mathcal{S}$ and the honest parties interact with $\mathcal{F}_{\mathsf{CN}}$ through a set of predefined interfaces, through which they submit their inputs to $\mathcal{F}_{\mathsf{CN}}$ and receive the corresponding outputs. By contrast, in the real world, we encounter a candidate protocol Π and an adversary $\mathcal{A}$ that corrupts the same set of parties as $\mathcal{S}$. The environment $\mathcal{Z}$ supplies the inputs of honest parties and reads their outputs, and in addition, interacts with $\mathcal{S}$ in the ideal world and $\mathcal{A}$ in the real world.

In this paper, we consider a *static corruption* model and assume that $\mathcal{S}$ and $\mathcal{A}$ corrupts at most f_0 and f_1 parties in $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively (f_0 and f_1 are the corruption thresholds tolerated by

DLT networks $\mathcal{N}_0$ and $\mathcal{N}_1$); we denote by $\mathcal{C}$ the set of corrupt parties.

The Ideal Functionality $\mathcal{F}_{\mathsf{CN}}$. Fig. 2 depicts ideal functionality $\mathcal{F}_{\mathsf{CN}}$, which is parametrized with networks $\mathcal{N}_0$ and $\mathcal{N}_1$. Each network exposes a CheckAuth interface that helps $\mathcal{F}_{\mathsf{CN}}$ check if a remote read request is authorized. This captures the property that access control is enforced by the honest parties in $\mathcal{N}_0$ and $\mathcal{N}_1$. In addition, $\mathcal{F}_{\mathsf{CN}}$ provides access to two interfaces RemoteRead and SendValue.

RemoteRead *Interface.* RemoteRead allows a party in $\mathcal{P}_{\bar{b}}$ to send a request to read the value of a key in $\mathcal{L}_b$, $b \in \{0,1\}$ and $\bar{b} = 1 - b$. The RemoteRead request carries a request identifier rid, identifies a key K in $\mathcal{L}_b$, and specifies a set of intended recipients $\mathcal{R}$. Upon such a request, $\mathcal{F}_{\mathsf{CN}}$ checks if the parties in $\mathcal{P}_0 \cap \mathcal{P}_1$ are all corrupt, and if so, waits for $\mathcal{S}$'s OK before transmitting the request to the parties in $\mathcal{P}_b$. This indicates the ability of a corrupt communication bridge to censor RemoteRead requests. Otherwise, if there is at least one honest party in $\mathcal{P}_0 \cap \mathcal{P}_1$, then $\mathcal{F}_{\mathsf{CN}}$ directly forwards the request to the parties in $\mathcal{P}_b$. Notice that during a RemoteRead invocation, $\mathcal{S}$ learns all the information contained in the request.

SendValue *Interface.* SendValue allows a party $\mathfrak{p}$ in $\mathcal{P}_b$ to send the value in $\mathcal{L}_b[\mathsf{K}]$ to a set of recipients $\mathcal{R} \subset \mathcal{P}_{\bar{b}}$, as a response to a RemoteRead request rid. If $\mathfrak{p}$ is honest, then $\mathcal{F}_{\mathsf{CN}}$ sends to $\mathcal{N}_b$ a CheckAuth request to verify whether the parties in $\mathcal{R}$ are authorized to read the value $\mathsf{v} \leftarrow \mathcal{L}_b[\mathsf{K}]$. If that's not the case, then $\mathcal{F}_{\mathsf{CN}}$ aborts. However, if $\mathfrak{p}$ is corrupt, then $\mathcal{F}_{\mathsf{CN}}$ skips the authorization check. Next, $\mathcal{F}_{\mathsf{CN}}$ checks if $\mathcal{P}_0 \cap \mathcal{P}_1 \subset \mathcal{C}$. If so, then $\mathcal{F}_{\mathsf{CN}}$ only forwards RemoteRead response to $\mathcal{R}$ after it receives $\mathcal{S}$'s go-ahead. If $\mathcal{P}_0 \cap \mathcal{P}_1 \not\subset \mathcal{C}$, then $\mathcal{F}_{\mathsf{CN}}$ directly distributes the RemoteRead response. The latter consists of tuple $(\mathsf{SendValue}, \mathsf{rid}, \mathsf{K}, \mathcal{L}_b, \mathfrak{p}, \mathsf{v})$. Providing the identity of the sender $\mathfrak{p}$ within the response reflects that the recipients authenticate the origin of the received value (i.e., the authentication property). In addition, if $\mathcal{R} \cap \mathcal{C} \neq \emptyset$, then $\mathcal{F}_{\mathsf{CN}}$ communicates the value v to $\mathcal{S}$. This signifies that if $\mathfrak{p}$ is honest, then $\mathcal{S}$ learns v only if one of the intended recipients is corrupt, capturing thus the confidentiality property.

We would like now to clarify why SendValue sends values from individual parties in $\mathcal{P}_b$, as opposed to a value from network $\mathcal{N}_b$ as a whole. Our rationale is two-fold: (1) DLT networks do not guarantee that all honest parties will store the same exact copy of the ledger at the same time. Instead they guarantee that within some time bound – which is known in the case of the synchronous setting and unknown in the case of partially-synchronous setting – the parties will store the same copy. (2) The recipients could decide themselves the policies that determine, depending on the sender, whether they trust a received value or not. For example, they could define policies that state that they will accept a value only if they receive it from $f_b + 1$ or $2f_b + 1$ parties, or from a specific party that they deem trustworthy.

Definition 8 (Secure Cross-Network Communication)**.** *A cross-network communication protocol* Π *securely realizes* $\mathcal{F}_{\mathsf{CN}}$*, if for any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *and any PPT environment* $\mathcal{Z}$*, there exists a PPT simulator* $\mathcal{S}$ *such that* $\textsc{real}_{\Pi,\mathcal{A},\mathcal{Z}} \approx_c \textsc{ideal}_{\mathcal{F}_{\mathsf{CN}},\mathcal{S},\mathcal{Z}}$*, where* $\textsc{real}_{\Pi,\mathcal{A},\mathcal{Z}}$ *is the random variable denoting the output of* $\mathcal{Z}$ *in the real world,* $\textsc{ideal}_{\mathcal{F}_{\mathsf{CN}},\mathcal{S},\mathcal{Z}}$ *is the random variable denoting the output of* $\mathcal{Z}$ *in the ideal world, and* $\approx_c$ *denotes computational indistinguishability.*

4 Realizing $\mathcal{F}_{\mathsf{CN}}$ using Fully Distributed Broadcast Encryption (FDBE)

In this section, we describe a cross-network communication protocol Π_{CN} that securely realizes the ideal functionality $\mathcal{F}_{\mathsf{CN}}$ using *fully distributed broadcast encryption* (FDBE).

4.1 Fully Distributed Broadcast Encryption (FDBE)

Let $\mathcal{P} = \{\mathfrak{p}_1, ..., \mathfrak{p}_n\}$ be a set of parties who wish to participate in an FDBE scheme. Each $\mathfrak{p}_i$ is deterministically assigned a broadcast slot (i.e., index) in $[n]$, and FDBE is defined by the following set of algorithms.

Public Parameters Initialization

$\mathsf{Init}(1^\kappa, n) \to \mathsf{pp}_0$: On input of security parameter κ and the size n of the set $\mathcal{P}$ supplied by some party in $\mathcal{P}$, Init outputs version 0 of the public parameters denoted pp_0.

$\mathsf{VerifyInit}(\mathsf{pp}_0, n) \to 0/1$: On input of pp_0, VerifyInit outputs 1 if it deems pp_0 well-formed, otherwise, it outputs 0.

Public Parameters Updates

$\mathsf{Update}(\mathsf{pp}_{i-1}, x) \to (\mathsf{pp}_i, \pi_i)$: Given version $i-1$ of the public parameters pp_{i-1} and a trapdoor x, Update returns version i of the public parameters, denoted pp_i, and a proof of correctness π_i. Hereafter, we assume that $\mathsf{Update}(\mathsf{pp}_{i-1}, \star)$ is called by the party assigned broadcast slot i.

$\mathsf{VerifyUpdate}(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i) \to 0/1$: On input of two consecutive versions of the public parameters pp_{i-1} and pp_i and a proof π_i, VerifyUpdate outputs 1 if π_i is deemed a valid proof demonstrating that pp_i is a correct update of pp_{i-1}; otherwise, it outputs 0. After everyone in $\mathcal{P}$ calls Update, the algorithms described next can be invoked.

Decryption Key Extraction

$\mathsf{ExtractDecKey}(\mathsf{pp}_n, x) \to \mathsf{dk}$: On input of public parameters pp_n, and trapdoor x of party $\mathfrak{p} \in \mathcal{P}$, ExtractDecKey returns $\mathfrak{p}$'s decryption key dk.

Broadcast Encryption and Decryption

$\mathsf{Encrypt}(m, \Sigma, \mathsf{pp}_n) \to \mathsf{ct}$: On input of a message m, a set $\Sigma \subset [n]$ identifying the broadcast slots of the potential recipients, and public parameter pp_n, Encrypt computes ciphertext ct.

$\mathsf{Decrypt}(\mathsf{ct}, \Sigma, \mathsf{dk}_i, \mathsf{pp}_n) \to \perp / m$: On input of ciphertext ct, a set $\Sigma \subset [n]$, a decryption key dk_i, and public parameters pp_n, Decrypt outputs either $\perp$ indicating that the decryption failed, or a message m signalling that the decryption succeeded.

Correctness and Security Definitions. We consider a static adversary $\mathcal{A}$ that corrupts parties before the assignment of broadcast slots and calling Init. An FDBE scheme is said to be secure if it satisfies the following security properties.

Correctness is twofold. It ensures that an honest execution of Init yield public parameters that will always be accepted by VerifyInit, whereas subsequent honest executions of Update will result

Adversary $\mathcal{A}$ begins by outputting a set $\mathcal{C} \subset \mathcal{P}$ of corrupt parties. We assume that $\mathcal{A}$ corrupts all parties except one that we refer to as party $\mathfrak{p}^*$ and that $\mathfrak{p}^*$ is assigned broadcast slot k.

Public Parameters Initialization. On behalf of one of the corrupt parties, $\mathcal{A}$ submits pp_0. The challenger accepts pp_0 only if $1 \leftarrow \mathsf{VerifyInit}(\mathsf{pp}_0, n)$.

Public Parameters Updates. $\mathcal{A}$ first submits a series of updated public parameters $(\mathsf{pp}_1, ..., \mathsf{pp}_{k-1})$ and the corresponding proofs $(\pi_1, ..., \pi_{k-1})$ to the challenger. The challenger accepts only if $\forall\ 1 \leq i < k : 1 \leftarrow \mathsf{VerifyUpdate}(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i)$. The challenger then calls $(\mathsf{pp}_k, \pi_k) \leftarrow \mathsf{Update}(\mathsf{pp}_{k-1}, x_k)$ on behalf of $\mathfrak{p}^*$, and outputs (pp_k, π_k) to $\mathcal{A}$. If $k < n$, then $\mathcal{A}$ is allowed to submit another series of updated public parameters $(\mathsf{pp}_{k+1}, ..., \mathsf{pp}_n)$ and proofs $(\pi_{k+1}, ..., \pi_n)$, and which the challenger accepts only if $\forall\ k+1 \leq i \leq n : 1 \leftarrow \mathsf{VerifyUpdate}(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i)$.

Challenge. On behalf of $\mathfrak{p}^*$, the challenger runs $\mathsf{dk}_k \leftarrow \mathsf{ExtractDecKey}(\mathsf{pp}_n, x_k)$. Next, $\mathcal{A}$ provides a message m and identifies a set of broadcast slots $\Sigma \subset [n]$. The challenger executes, accordingly, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(m, \Sigma, \mathsf{pp}_n)$ and $\mathsf{out} \leftarrow \mathsf{Decrypt}(\mathsf{ct}, \Sigma, \mathsf{dk}_k, \mathsf{pp}_n)$.

$\mathcal{A}$ succeeds in this experiment **iff** $k \in \Sigma$ and $\mathsf{out} \neq m$.

Figure 3: Key Extractability Experiment for FDBE

Adversary $\mathcal{A}$ begins by outputting a set $\mathcal{R} \subset \mathcal{P}$. We assume that $\mathcal{A}$ corrupts all parties except $\mathcal{R}$.

Public Parameters Initialization. On behalf of one of the corrupt parties $\mathcal{A}$ submits pp_0. The challenger accepts pp_0 only if $1 \leftarrow \mathsf{VerifyInit}(\mathsf{pp}_0, n)$.

Public Parameters Updates. $\mathcal{A}$ and the challenger engage in a series of interleaved calls to Update such that $\mathcal{A}$ makes the calls on behalf of the corrupt parties whereas the challenger makes the calls on behalf of the honest parties $\mathcal{R}$. This phase concludes by outputting a set of public parameters pp_n.

Challenge. $\mathcal{A}$ sends two messages m_0 and m_1 to the challenger. The challenger randomly picks $b \in \{0, 1\}$ and outputs $\mathsf{ct}_b = \mathsf{Encrypt}(m_b, \Sigma, \mathsf{pp}_n)$ to $\mathcal{A}$, where Σ is the set of broadcast slots assigned to the parties in $\mathcal{R}$.

Guess. $\mathcal{A}$ outputs a guess b^* of b. $\mathcal{A}$ succeeds in this experiment **iff** $b = b^*$.

Figure 4: Confidentiality Experiment for FDBE

in public parameters that will always be accepted by VerifyUpdate. More formally, the following equalities hold:

$$\Pr[\mathsf{VerifyInit}(\mathsf{pp}_0, n) \rightarrow 1\ \wedge\ \forall 1 \leq i \leq l : \mathsf{VerifyUpdate}(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i) \rightarrow 1 | \\ \mathsf{pp}_0 \leftarrow \mathsf{Init}(1^\kappa, n)\ \wedge\ \forall 1 \leq i \leq l : (\mathsf{pp}_i, \pi_i) \leftarrow \mathsf{Update}(\mathsf{pp}_{i-1}, x_i)] = 1$$

It also guarantees that after an honest execution of Init followed by consecutive honest executions of Update by all the parties, the honest parties will always be able to extract their decryption keys and decrypt relevant ciphertexts by calling ExtractDecKey and Decrypt respectively. In other words, for any party with broadcast slot in set Σ and which has previously called Update with some trapdoor

x, the following holds:

$$\Pr[\mathsf{Decrypt}(\mathsf{ct}, \Sigma, \mathsf{dk}, \mathsf{pp}_n) \to m \mid \mathsf{ct} \leftarrow \mathsf{Encrypt}(m, \Sigma, \mathsf{pp}_n) \wedge \mathsf{dk} \leftarrow \mathsf{ExtractDecKey}(\mathsf{pp}_n, x)] = 1$$

Security is also twofold in the sense that we require the following (informal) properties to hold:

- *Key extractability* captures the property that corrupt parties cannot prevent honest parties from computing their decryption keys and correctly decrypting well-formed ciphertexts that are intended for them. More formally, we say that a FDBE scheme guarantees key extractability **iff**, for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the experiment depicted in Figure 3 is negligible.
- *Confidentiality* guarantees that for any ciphertext ct generated as $\mathsf{ct} \leftarrow \mathsf{Encrypt}(m, \Sigma, \mathsf{pp}_n)$ for any (arbitrarily chosen) message m and any subset $\Sigma \subset [n]$, only the parties in Σ can successfully decrypt ct and obtain m. We consider chosen-plaintext security and formalize confidentiality using the experiment described in Figure 4. Let $q_{\kappa,\mathcal{A}}$ be the probability that $\mathcal{A}$ succeeds in the experiment, and $\epsilon_{\kappa,\mathcal{A}}$ be the value $q_{\kappa,\mathcal{A}} - 1/2$. We say that a FDBE scheme is secure against chosen-plaintext attacks *in the static corruption setting*, **iff** for any PPT adversary $\mathcal{A}$, the value $\epsilon_{\kappa,\mathcal{A}}$ as defined above is negligible.

4.2 Protocol Realizing $\mathcal{F}_{\mathsf{CN}}$

In this subsection, we describe a cross-network communication protocol Π_{CN} that securely realizes the ideal functionality $\mathcal{F}_{\mathsf{CN}}$ by using any FDBE scheme in a fully black-box manner.

Let $\mathcal{N}_0$ and $\mathcal{N}_1$ be two networks operated by parties $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively. As participants in permissioned blockchains, parties in $\mathcal{P}_0$ and $\mathcal{P}_1$ are endowed each with a pair of public and secret keys. We assume that $\mathcal{N}_0$ and $\mathcal{N}_1$ expose, in addition to CheckAuth, two interfaces Deploy and Execute that enable the deployment and execution of smart contracts. The protocol Π_{CN} comprises the following phases.

Setup. Parties in $\mathcal{P}_0$ and $\mathcal{P}_1$ are first assigned broadcast slots in their respective networks. Next, for $b \in \{0, 1\}$, let SC_b be a smart contract that (i) stores the public keys of the parties in $\mathcal{P}_b$; (ii) implements the logic of VerifyInit and VerifyUpdate of an instantiation of a distributed BE scheme denoted FDBE_b; and (iii) enforces that a party in $\mathcal{P}_b$ updates the public parameters of FDBE_b at most once, and in the order prescribed by the broadcast slots. Parties in $\mathcal{P}_b$ send message $(\mathsf{Deploy}, \mathsf{SC}_b)$ to deploy SC_b on network $\mathcal{N}_b$, and conclude the setup if the smart contract deployment succeeds.

Public Parameters Initialization. A party $\mathfrak{p} \in \mathcal{P}_b$ first calls $\mathsf{Init}(1^\kappa, n_b) \to \mathsf{pp}_{b,0}$. She then prepares a signed Init transaction $\mathsf{itx} = (\mathsf{SC}_b, \mathsf{Init}, \mathsf{pp}_{b,0}, \mathsf{pk}, \sigma)$, such that pk is her public key and σ is her signature on $(\mathsf{SC}_b, \mathsf{Init}, \mathsf{pp}_{b,0})$. Finally, $\mathfrak{p}$ sends message $(\mathsf{Execute}, \mathsf{itx})$ to $\mathcal{N}_b$. This message executes SC_b on input $(\mathsf{Init}, \mathsf{pp}_{b,0}, \mathsf{pk}, \sigma)$. SC_b consequently checks if: this is the first Init transaction, pk is the public key of a party in $\mathcal{P}_b$, σ is a valid signature on $(\mathsf{SC}_b, \mathsf{Init}, \mathsf{pp}_{b,0})$ relative to pk, and $1 \leftarrow \mathsf{VerifyInit}(\mathsf{pp}_{b,0}, n_b)$. If any of these checks fails, then SC_b rejects the initialization of the public parameters. Otherwise, it adds entry $\mathcal{L}_b[\mathsf{FDBE.PP}] \leftarrow \mathsf{pp}_{b,0}$ and $\mathcal{L}_b[\mathsf{FDBE.Update}] \leftarrow \emptyset$.

Public Parameters Updates. Party $\mathfrak{p} \in \mathcal{P}_b$, assigned broadcast slot i, first fetches the current public parameters by reading entry $\mathcal{L}_b[\mathsf{FDBE.PP}] \to \mathsf{pp}_{b,i-1}$, then invokes $(\mathsf{pp}_{b,i}, \pi_{b,i}) \leftarrow \mathsf{Update}(\mathsf{pp}_{b,i-1}, x)$, and prepares Update transaction $\mathsf{utx} = (\mathsf{SC}_b, \mathsf{Update}, \mathsf{pp}_{b,i}, \pi_{b,i}, \mathsf{pk}, \sigma)$, where

pk is the public key of $\mathfrak{p}$ and σ is a signature on tuple $(\mathsf{SC}_b, \mathsf{Update}, \mathsf{pp}_{b,i}, \pi_{b,i})$ using the corresponding secret key sk. $\mathfrak{p}$ concludes by sending message $(\mathsf{Execute}, \mathsf{utx})$ to $\mathcal{N}_b$. This message triggers SC_b's execution on input $(\mathsf{Update}, \mathsf{pp}_{b,i}, \pi_{b,i}, \mathsf{pk}, \sigma)$, which entails checking *all of the following*: (i) pk is the public key of a party in $\mathcal{P}_b$; (ii) $\mathsf{pk} \notin \mathcal{L}_b[\mathsf{FDBE.Update}]$ (i.e., this is the first update from party $\mathfrak{p}$); (iii) party associated with pk is assigned the broadcast slot that matches the *current* update; (iv) $\mathsf{VerifyUpdate}(\mathcal{L}_b[\mathsf{FDBE.PP}], \mathsf{pp}_{b,i}, \pi_{b,i}) \rightarrow 1$; (v) and σ is a valid signature on $(\mathsf{SC}_b, \mathsf{Update}, \mathsf{pp}_{b,i}, \pi_{b,i})$ relative to pk. If any of these checks fails, then SC_b rejects the update. Otherwise, it updates entries $\mathcal{L}_b[\mathsf{FDBE.PP}] \leftarrow \mathsf{pp}_{b,i}$ and $\mathcal{L}_b[\mathsf{FDBE.Update}] \leftarrow \mathcal{L}_b[\mathsf{FDBE.Update}] \cup \mathsf{pk}$.

Decryption Key Extraction. Using her trapdoor x, party $\mathfrak{p} \in \mathcal{P}_b$ retrieves her decryption key by invoking $\mathsf{ExtractDecKey}$ with input (pp_{b,n_b}, x).

Remote Reads. We assume that the public parameters pp_{b,n_b} and the broadcast slots $\mathcal{P}_b$ are transmitted to $\mathcal{P}_{\bar{b}}$. A party $\mathfrak{p} \in \mathcal{P}_b$ sends through the communication bridge $\mathcal{P}_0 \cap \mathcal{P}_1$ a signed $\mathsf{RemoteRead}$ request to read the value of a key K in $\mathcal{L}_{\bar{b}}$, on behalf of a group of recipients $\mathcal{R} \subset \mathcal{P}_b$. The $\mathsf{RemoteRead}$ request correspondingly consists of tuple $(\mathsf{RemoteRead}, \mathsf{rid}, \mathcal{R}, \mathsf{K}, \mathsf{pk}, \sigma)$, where rid is the unique session identifier of the request, pk is the public key of $\mathfrak{p}$ and σ is a signature on $(\mathsf{RemoteRead}, \mathsf{rid}, \mathcal{R}, \mathsf{K})$. While the method through which this request is distributed to the parties in $\mathcal{P}_{\bar{b}}$ is out of scope, it can be accommodated by using $\mathcal{N}_{\bar{b}}$ as a broadcast channel. Basically, a party in $\mathcal{P}_0 \cap \mathcal{P}_1$ submits a transaction that includes the $\mathsf{RemoteRead}$ request. Upon seeing the request, a party $\mathfrak{p}' \in \mathcal{P}_{\bar{b}}$ checks if: the party with public key pk is authorized to issue $\mathsf{RemoteRead}$ requests; the recipients in $\mathcal{R}$ are allowed to read the value stored at key K; and σ is a valid signature on tuple $(\mathsf{RemoteRead}, \mathsf{rid}, \mathcal{R}, \mathsf{K})$ under public key pk. If all the checks succeed, then $\mathfrak{p}'$ (i) fetches from her local copy of the ledger value $\mathsf{v} \leftarrow \mathcal{L}_{\bar{b}}[\mathsf{K}]$; (ii) calls $\mathsf{Encrypt}(\mathsf{v}, \Sigma, \mathsf{pp}_{b,n_b}) \rightarrow \mathsf{ct}$ (whereby Σ is the set of broadcast slots of the parties identified in $\mathcal{R}$); (iv) computes a signature σ' on $(\mathsf{rid}, \mathcal{R}, \mathsf{K}, \mathsf{ct})$ using her secret key sk'; (v) and sends through the communication bridge tuple $(\mathsf{rid}, \mathcal{R}, \mathsf{K}, \mathsf{ct}, \mathsf{pk}', \sigma')$, where pk' is her public key. The communication bridge then distributes this tuple to the relevant parties by submitting a transaction to $\mathcal{N}_b$. On seeing the tuple, a party $\mathfrak{p} \in \mathcal{R}$ verifies if: (i) she partook in a remote read for key K with session identifier rid; (ii) pk' is the public key of a party in $\mathcal{P}_{\bar{b}}$; (iii) and σ' is a valid signature on $(\mathsf{rid}, \mathcal{R}, \mathsf{K}, \mathsf{ct})$ relative to pk'. If all checks succeed, $\mathfrak{p}$ calls $\mathsf{v} \leftarrow \mathsf{Decrypt}(\mathsf{ct}, \Sigma, \mathsf{dk}, \mathsf{pp}_{b,n_b})$.

Theorem 4.1 (Security of Π_{CN})**.** *Assuming that: (i) the underlying FDBE scheme satisfies correctness, key extractability and confidentiality, and (ii) the digital signature scheme satisfies existential unforgeability under chosen message attacks, the above protocol* Π_{CN} *securely realizes* $\mathcal{F}_{\mathsf{CN}}$.

Proof Sketch. We prove Theorem 4.1 using a hybrid argument, where the sequence of hybrids is as follows.

Hybrid 0. In this game, $\mathsf{RemoteRead}$ requests between $\mathcal{N}_0$ and $\mathcal{N}_1$ are served by ideal functionality F.

Hybrid 1. This game is similar to **Hybrid 0** except that the honest participants in $\mathcal{N}_0$ and $\mathcal{N}_1$ deploy smart contracts SC_0 and SC_1.

Hybrid 2. This game is similar to **Hybrid 1** except that parties in $\mathcal{N}_0$ and $\mathcal{N}_1$ submit transactions to initialize public parameters $\mathsf{pp}_{0,0}$ and $\mathsf{pp}_{1,0}$.

Hybrid 3. This game is similar to **Hybrid 1** except that parties in $\mathcal{N}_0$ and $\mathcal{N}_1$ submit transactions to update the public parameters. Without loss of generality, we assume that all parties in $\mathcal{P}_0$ and $\mathcal{P}_1$ submitted update requests.

Hybrid 4. In this game, we replace RemoteRead calls to $\mathcal{F}_{\mathsf{CN}}$ with Π_{CN}'s RemoteRead requests. Note that if there exists an honest party in the communication bridge, then $\mathcal{F}_{\mathsf{CN}}$ transmits RemoteRead request along with the identity of the sender to the parties in the destination network - after some delay imposed by $\mathcal{S}$. Otherwise, $\mathcal{F}_{\mathsf{CN}}$ forwards the request only if $\mathcal{S}$ agrees. Turning to Π_{CN}, we recall that an honest party in the bridge will submit any RemoteRead request it receives for to its destination network. After some delay, the request will be disseminated to all the relevant parties, by submitting a transaction that contains the request to the network. If all the parties in the bridge are corrupt, then they can block the RemoteRead requests if they choose to. This illustrates that a request that's communicated when interacting with $\mathcal{F}_{\mathsf{CN}}$ will be communicated during an execution of Π_{CN}, and vice-versa.

When interacting with $\mathcal{F}_{\mathsf{CN}}$, $\mathcal{S}$ learns the identity of the sender, the identifier of the request, the set of potential recipients, and the key to be read, and an execution of Π_{CN} reveals exactly the same information. We recall that a RemoteRead request in Π_{CN} carries the request identifier, the public key of the sender, a signature under that public key, the key to be read and the set of potential recipients. Since the signature is *existentially unforgeable*, then the signature with the public key will successfully identify the sender of the request.

Therefore, replacing calls to the RemoteRead interface exposed by $\mathcal{F}_{\mathsf{CN}}$ by an execution of Π_{CN} will not impact the view of the adversary, and consequently, **Hybrid 4** is indistinguishable from **Hybrid 3**.

Hybrid 5. This game is similar to **Hybrid 4** with the only difference being replacing calls to SendValue by RemoteRead responses that follows the specifications of Π_{CN}.

During a call to $\mathcal{F}_{\mathsf{CN}}$ using SendValue, $\mathcal{F}_{\mathsf{CN}}$ performs access control checks only if the sender is honest. During an execution of Π_{CN}, corrupt senders can violate access control checks and share data with unauthorized parties. When the sender is honest, both $\mathcal{F}_{\mathsf{CN}}$ and Π_{CN} enforce access control: $\mathcal{F}_{\mathsf{CN}}$ by calling the CheckAuth interface, and Π_{CN} by counting on the honest sender to follow the protocol and abide by her network's rules. Similar to the argument in **Hybrid 4**, a response that's communicated when interacting with $\mathcal{F}_{\mathsf{CN}}$ will be communicated during an execution of Π_{CN}, and vice-versa.

Now we show that when an honest sender uses Π_{CN} to send her response, the response does not leak any information beyond what a call to $\mathcal{F}_{\mathsf{CN}}$'s interface RemoteRead leaks.

Notice that if one of the recipients is corrupt, then $\mathcal{S}$ (interacting with $\mathcal{F}_{\mathsf{CN}}$) learns the set of recipients, the identity of the sender, the identifier of a RemoteRead request, and the key for which the said request was issued, and the plaintext value. Therefore $\mathcal{S}$ can successfully simulate the honest sender for the corrupt recipient: that is, $\mathcal{S}$ computes a broadcast encryption of the plaintext value intended for the designated recipients, and the signature on the RemoteRead response. Note that given the key extractability property of FDBE, the recipients will be able to successfully decrypt the ciphertext using their decryption keys and receive the plaintext value encrypted by $\mathcal{S}$.

If all the recipients are honest, then $\mathcal{S}$ only learns all the previously-mentioned information except the plaintext value. $\mathcal{S}$ simulates the response of the honest sender by computing a broadcast encryption of a randomly-chosen value for the honest recipients, and then signing the corresponding response. Note that thanks to the confidentiality (IND-CPA) property of the FDBE, one cannot tell whether a ciphertext encrypts a random value or the value of given key in the target ledger. This demonstrates that **Hybrid 5** is indistinguishable from **Hybrid 4**. □

4.3 Bilinear Pairing-based Construction of FDBE

In this section, we present a FDBE scheme based on bilinear pairings that supports constant-sized keys and ciphertexts, and *fully distributed* public parameters setup and key generation. This yields a concrete instance of our protocol Π_{CN} from Section 4.2. The starting point of our design is the bilinear pairing-based BE scheme with constant-size decryption keys and constant-size ciphertexts from [BGW05]. The original scheme from [BGW05] relies on a trusted (centralized) setup and a trusted key generation procedure (where all participants must contact a centrally trusted entity holding the master secret key to obtain their individual decryption keys). As a result, any compromise of this central trusted entity completely breaks the security of the scheme. In our design of FDBE, we augment this scheme to achieve decentralized setup and distributed key generation, without relying on any central trusted party. Unlike recent BE constructions with distributed key generation [KMW23, GKPW24, CGPP24, CGPW25] that require trusted setup to generate a common reference string, our scheme totally avoids any (centralized) trust assumptions.

Notations. Let $\mathcal{P} = \{\mathfrak{p}_1, ..., \mathfrak{p}_n\}$ be a set of parties who wish to participate in a FDBE scheme. Let $\mathbb{G}$ be a group of prime order p that admits a non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and let P and Q be two uniformly-random generators for $\mathbb{G}$ (see Section 2.1 for background material on pairings). Let $[n]$ denote the set of integers $\{1, ..., n\}$ and let $[n, m]$ denote the set of integers $\{n, ..., m\}$. We use upper-case letters to refer to elements in $\mathbb{G}$, whereas lower-case letters are used to refer to elements in $\mathbb{F}_p$. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\kappa$ and $\mathcal{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ be cryptographic hash functions.

Building Blocks. We use the following primitives as building blocks (see Section 2.2 for the formal definitions): (i) an IND-CPA secure symmetric-key encryption (SKE) scheme $(\mathcal{E}, \mathcal{D})$ with κ-bit secret keys, and (ii) a non-interactive zero-knowledge (NIZK) argument of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ that satisfies knowledge-extraction soundness and zero knowledge (in the random oracle model). In particular, the NIZK argument of knowledge is used to generate proofs for instance-witness pairs of the form $(\mathbb{x}, \mathbb{w}) = ((P, P'), x)$ where $P, P' \in \mathbb{G}$ and $x \in \mathbb{F}_p$ satisfying the discrete log relation:

$$P' = P^x \tag{4.1}$$

The Construction. We now present the detailed construction. A summary of the construction appears in Fig. 5.

Setup. During setup, each party $\mathfrak{p}_i$ calls $\mathcal{H}$ on each party identifier, and orders the resulting hashes lexicographically. The broadcast slot assigned to $\mathfrak{p}_i$ corresponds to the placement of the hash of her identifier in the lexicographic order.

Public Parameters Initialization

$\mathsf{Init}(1^\kappa, n) \to \mathsf{pp}_0$: On input security parameter κ and the total number n of potential participants, Init computes $P = \mathcal{H}_{\mathbb{G}}(0)$ and $Q = \mathcal{H}_{\mathbb{G}}(1)$, and outputs pp_0, which is defined as:

$$\mathsf{pp}_0 = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n) \text{ where } P_j = P \text{ and } \ T_j = Q$$

Observe that the generation of pp_0 does not involve any secrets.

$\mathsf{VerifyInit}(\mathsf{pp}_0, n) \to 0/1$: On input $\mathsf{pp}_0 = (P, \{P_j\}_{j \in [2n] \setminus \{n+1\}}, Q, \{T_j\}_{j \in [n]})$ $\mathsf{VerifyInit}$ outputs 1 if *all of the following hold*: (i) $P = \mathcal{H}_{\mathbb{G}}(0)$, (ii) $Q = \mathcal{H}_{\mathbb{G}}(1)$, (iii) $P_i = P$ for each $i \in [2n] \setminus n+1$, and (iv) $T_j = Q$ for each $j \in [n]$[2]. Otherwise, it outputs 0.

[2] P and Q can also be computed using $\mathcal{H}_{\mathbb{G}}$ and any two seeds. The seeds though needs to be transmitted along with pp_0.

Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a non-degenerate bilinear map (where $\mathbb{G}$ is a group of prime order p). Let $(\mathcal{E}, \mathcal{D})$ be a semantically-secure symmetric encryption scheme with key space $\mathcal{K}$. Let Sign be a signature algorithm and Ver the corresponding verification algorithm. Let $\mathcal{H} : \{0,1\}^* \to \mathcal{K}$ and $\mathcal{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ be two hash functions (modeled as a random oracle).

Setup: During setup, each party $\mathfrak{p}_i \in \mathcal{P}$ calls $\mathcal{H}$ on the identifiers of all the parties in $\mathcal{P}$. $\mathfrak{p}_i$ then orders the computed hashes η_j lexicographically. This results in a permutation τ of the hashes defined as $j \to \tau(j) : \eta_{\tau(1)} < \eta_{\tau(2)} < ... < \eta_{\tau(n)}$, and the broadcast slot assigned to $\mathfrak{p}_i$ is $\tau(i)$.

$\mathsf{Init}(1^\kappa, n) \to \mathsf{pp}_0$: On input the security parameter κ and an upper bound n of the total number of potential recipients of broadcast ciphertexts, Init computes $P = \mathcal{H}_{\mathbb{G}}(0)$ and $Q = \mathcal{H}_{\mathbb{G}}(1)$, generates and outputs

$$\mathsf{pp}_0 = (P, P_1, \ldots, P_n, P_{n+2}, \ldots, P_{2n}, Q, T_1, \ldots, T_n)$$

where $P_j = P$ for each $j \in [2n]$ and $U_j = Q$ for each $j \in [n]$.

$\mathsf{VerifyInit}(\mathsf{pp}_0, n) \to 0/1$: On input $\mathsf{pp}_0 = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$, $\mathsf{VerifyInit}$ outputs 1 if **(1)** $P = \mathcal{H}_{\mathbb{G}}(0)$, **(2)** $Q = \mathcal{H}_{\mathbb{G}}(1)$, **(3)** $P_j = P$ for each $j \in [2n] \setminus \{n+1\}$, and **(4)** $T_j = Q$ for each $j \in [n]$. Otherwise, it outputs 0.

$\mathsf{Update}(\mathsf{pp}_{i-1}, x_i) \to (\mathsf{pp}_i, \pi_i)$: Given $\mathsf{pp}_{i-1} = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$ and a trapdoor $x_i \in \mathbb{F}_p$ provided the party assigned broadcast slot i, Update computes $\mathsf{pp}_i = (P, P'_1, ..., P'_n, P'_{n+2}, ..., P'_{2n}, Q, T'_1, ..., T'_n)$ where $P'_j = P_j^{x_i^j} \quad \forall j \in [2n] \setminus \{n+1\}$; $T'_i = T_i$; $T'_j = T_j^{x_i^j} \quad \forall j \in [n] \setminus \{i\}$. Update also produces a non-interactive zero-knowledge proof π_i for the following relation: $\exists x_i \in \mathbb{F}_p : P'_1 = P_1^{x_i}$. Finally, Update outputs (pp_i, π_i).

$\mathsf{VerifyUpdate}(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i) \to 0/1$: On input $\mathsf{pp}_{i-1} = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$, $\mathsf{pp}_i = (P, \{P'_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T'_j\}_{j\in[n]})$, and proof π_i, $\mathsf{VerifyUpdate}$ checks if **(1)** π_i is a valid zero-knowledge proof for the relation depicted in 4.1, **(2)** $T'_i = T_i$ and **(3)** the following equality holds for randomly chosen $(\alpha, \beta) \in \mathbb{F}_p^2$:

$$e(P'_1, \prod_{j=1}^{n-1} {P'_j}^{\alpha^j} \prod_{j=n+2}^{2n-1} {P'_j}^{\alpha^j}) e({P'_2}^{\beta}, P'_n) = e(P, {P'_{n+2}}^{\beta} \prod_{j=1}^{n-1} {P'_{j+1}}^{\alpha^j} \prod_{j=n+2}^{2n-1} {P'_{j+1}}^{\alpha^j})$$

If any of these checks fails, then $\mathsf{VerifyUpdate}$ rejects and outputs 0; otherwise it accepts and outputs 1.

$\mathsf{ExtractDecKey}(\mathsf{pp}_n, x_i) \to \mathsf{dk}_i$: On input $\mathsf{pp}_n = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$, and x_i, $\mathsf{ExtractDecKey}$ returns $\mathsf{dk}_i = T_i^{x_i^i}$.

$\mathsf{Encrypt}(m, \Sigma, \mathsf{pp}_n) \to C$: On input of a message m, a set $\Sigma \subset [n]$ that identifies the broadcast slots of potential recipients, and public parameters $\mathsf{pp}_n = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$, $\mathsf{Encrypt}$ outputs ciphertext $\mathsf{ct} = (h, C)$, where

$$h = \left(P^r, \left(Q \cdot \prod_{j\in\Sigma} P_{n+1-j} \right)^r \right) \; ; \Omega = e(P_1^r, P_n) \; ; \; K = \mathcal{H}(\Omega) \; ; \; C = \mathcal{E}(K, m)$$

$\mathsf{Decrypt}(\mathsf{ct}, \Sigma, \mathsf{dk}_i, \mathsf{pp}_n) \to \perp / m$: On input of ciphertext $\mathsf{ct} = (h = (h_0, h_1), C)$, a set $\Sigma \subset [n]$, a decryption key dk_i such that $i \in \Sigma$, and public parameters $\mathsf{pp}_n = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$, $\mathsf{Decrypt}$ computes

$$P^* = \prod_{j\in\Sigma, j\neq i} P_{n+1-j+i} \; ; \; \Omega' = e\left(P_i, h_1\right) / e\left(h_0, \mathsf{dk}_i \cdot P^*\right)$$

and outputs $m' = \mathcal{D}(\mathcal{H}(\Omega'), C)$.

Figure 5: FDBE from Bilinear Pairing Groups (Summary of Construction from Section 4.3)

Public Parameters Updates

$\mathsf{Update}(\mathsf{pp}_{i-1}, x_i) \to (\mathsf{pp}_i, \pi_i)$: Given pp_{i-1} and a trapdoor $x_i \in \mathbb{F}_p$ from the party assigned the ith

slot, Update first parses $\mathsf{pp}_{i-1} := (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$ and computes

$$\mathsf{pp}_i = (P, P'_1, ..., P'_n, P'_{n+2}, ..., P'_{2n}, Q, T'_1, ..., T'_n)$$
$$P'_j = P_j^{x_i^j} \ ; \ T'_i = T_i \ ; \ \ \forall j \neq i : T'_j = T_j^{x_i^j}$$

Update also produces a NIZK proof $\pi_i \leftarrow \mathcal{P}((P'_1, P_1), x_i)$ for the relation in Eq. 4.1. Finally, Update outputs (pp_i, π_i).

$\mathsf{VerifyUpdate}(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i) \rightarrow 0/1$: On input pp_{i-1}, pp_i and proof π_i, VerifyUpdate first parses

$$\mathsf{pp}_{i-1} := (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n)$$
$$\mathsf{pp}_i := (P, P'_1, ..., P'_n, P'_{n+2}, ..., P'_{2n}, Q, T'_1, ..., T'_n).$$

Then, it verifies if **(1)** $\mathcal{V}((P'_1, P_1), \pi_i) = 1$, **(2)** $T'_i = T_i$ and **(3)** the following equations hold

$$j \notin \{n, n+1\} : e(P'_1, P'_j) = e(P, P'_{j+1}) \tag{4.2}$$
$$e(P'_2, P'_n) = e(P, P'_{n+2}) \tag{4.3}$$
$$j \neq i : e(T'_j, P_j) = e(T_j, P'_j)$$

If any of these checks fails, then VerifyUpdate rejects and outputs 0; otherwise, it accepts and outputs 1. Notice that thanks to the Schwartz-Zippel lemma, the verification equations in 4.2 and equation 4.3 can be aggregated into one equation. Actually, if VerifyUpdate randomly chooses $(\alpha, \beta) \in \mathbb{F}_p^2$ and the following equality holds:

$$e(P'_1, \prod_{j=1}^{n-1} {P'_j}^{\alpha^j} \prod_{j=n+2}^{2n-1} {P'_j}^{\alpha^j}) e({P'_2}^{\beta}, P'_n) = e(P, {P'}^{\beta}_{n+2} \prod_{j=1}^{n-1} {P'}^{\alpha^j}_{j+1} \prod_{j=n+2}^{2n-1} {P'}^{\alpha^j}_{j+1})$$

then VerifyUpdate can conclude that equations 4.2 and 4.3 hold with overwhelming probability $1 - (2n-1)/p$. This aggregation reduces the number of pairings required for the verification from $6n - 4$ pairings to $2n + 1$.

We note that if $\mathfrak{p}_i$ misses her round of update, then the protocol moves onto the next round, and $\mathfrak{p}_i$ is removed from the broadcast group. We argue that this is fair, especially, if the ledger is *censorship resistant*, which guarantees that the transactions of honest parties will always be executed.

After everyone in $\mathcal{P}$ successfully updates the public parameters, the algorithms below are invoked.

Decryption Key Extraction and Broadcast Encryption

$\mathsf{ExtractDecKey}(\mathsf{pp}_n, x_i) \rightarrow \mathsf{dk}_i$: On input public parameters $\mathsf{pp}_n = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$, and the trapdoor x_i of the party assigned slot i, ExtractDecKey returns decryption key $\mathsf{dk}_i = T_i^{x_i^i}$.

$\mathsf{Encrypt}(m, \Sigma, \mathsf{pp}_n) \rightarrow \mathsf{ct}$: On input a message m, a set $\Sigma \subset [n]$ identifying the broadcast slots of the recipients, and public parameters pp_n, Encrypt outputs ciphertext $\mathsf{ct} = (h, C)$, where

$$h = \left(P^r, \left(Q \cdot \prod_{j\in\Sigma} P_{n+1-j}\right)^r\right) \ ;$$
$$\Omega = e(P_1^r, P_n) \ ; \ K = \mathcal{H}(\Omega) \ ; \ C = \mathcal{E}(K, m)$$

$\mathsf{Decrypt}(\mathsf{ct}, \Sigma, \mathsf{dk}_i, \mathsf{pp}_n) \rightarrow \perp$ /m: On input of ciphertext $\mathsf{ct} = (h = (h_0, h_1), C)$, a set $\Sigma \subset [n]$ identifying the broadcast slots of the recipients, a decryption key dk_i, and public parameters pp_n, $\mathsf{Decrypt}$ computes

$$P^* = \prod_{j \in \Sigma, j \neq i} P_{n+1-j+i} \; ;$$
$$\Omega' = e\left(P_i, h_1\right) \Big/ e\left(h_0, \mathsf{dk}_i \cdot P^*\right)$$

and outputs $m' = \mathcal{D}(\mathcal{H}(\Omega'), C)$.

Correctness and Security. Correctness follows immediately from the correctness of $(\mathcal{E}, \mathcal{G})$, whereas key extractability follows from the equations verified during $\mathsf{VerifyInit}$ and $\mathsf{VerifyUpdate}$. Finally, confidentiality is assured under the *augmented n-bilinear Diffie-Hellman exponent* (n-BDHE) assumption, which we state below. This assumption is an augmented version of the n-BDHE assumption that was used to prove security of the original BE scheme from [BGW05]. For completeness, we recall both assumptions.

The n-BDHE Assumption [BGW05]. We first recall the n-BDHE assumption that was used to prove security of the original BE scheme from [BGW05].

Definition 9 (n-BDHE Assumption [BGW05])**.** *Let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a non-degenerate bilinear map (where $\mathbb{G}$ is a group of prime order p, parameterized by the security parameter κ), let P and H be uniformly random generators for $\mathbb{G}$, let $x \leftarrow \mathbb{F}_p$, let $\{P_i = P^{x^i}\}_{i \in [1,2n] \setminus \{n+1\}}$, and let*

$$\Gamma_0 = e(P, H)^{x^{n+1}} \; ; \; \Gamma_1 \leftarrow \mathbb{G}_T.$$

For any security parameter $\kappa \in \mathbb{N}$, any PPT algorithm $\mathcal{A}$, and any bit $b \in \{0, 1\}$, define the probability $q_{\kappa,\mathcal{A},b}$ as

$$q_{\kappa,\mathcal{A},b} := \Pr\left[\mathcal{A}\left(P, \{P_i\}_{i \in [1,n] \setminus \{n+1\}}, H, \Gamma_b\right) = 0\right].$$

The n-BDHE assumption states that for any security parameter κ and for any PPT algorithm $\mathcal{A}$, we have

$$|q_{\kappa,\mathcal{A},0} - q_{\kappa,\mathcal{A},1}| \leq \mathsf{negl}(\kappa)$$

The Augmented n-BDHE Assumption. We now present the augmented n-BDHE assumption, which augments the n-BDHE assumption by allowing the adversary to access some additional terms.

Definition 10 (Augmented n-BDHE Assumption)**.** *Let $\mathbb{G}$ be a group of prime order $p = O(2^\kappa)$ (κ being the security parameter) that admits a non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and let P and H be uniformly random generators for $\mathbb{G}$. Let $x_1, ..., x_n \leftarrow \mathbb{F}_p$ and $x = \prod_{i=1}^n x_i$, and let*

$$\{P_i = P^{x^i}\}_{i \in [2n] \setminus \{n+1\}} \; ; \; \{T_{i,j} = P_i^{1/x_j^j}\}_{i \in [2n], j \in [n]}$$
$$\Gamma_0 = e(P, H)^{x^{n+1}} \; ; \; \Gamma_1 \leftarrow \mathbb{G}_T.$$

For any PPT algorithm $\mathcal{A}$ and any bit $b \in \{0, 1\}$, define the probability $q'_{\kappa,\mathcal{A},b}$ as the following probability expression

$$q'_{\kappa,\mathcal{A},b} = \Pr\left[\mathcal{A}\left(P, \{P_i\}_{i \in [2n] \setminus \{n+1\}}, \{T_{i,j}\}_{i \in [2n], j \in [n]}, H, \Gamma_b\right) = 0\right]$$

The augmented n-BDHE assumption states that for any security parameter κ and for any PPT algorithm $\mathcal{A}$,

$$\left|q'_{\kappa,\mathcal{A},0} - q'_{\kappa,\mathcal{A},1}\right| \leq \mathsf{negl}(\kappa)$$

Theorem 4.2 (Correctness and Security of FDBE Scheme)**.** *Our proposed FDBE scheme satisfies correctness assuming that* $(\mathcal{E}, \mathcal{D})$ *satisfies correctness and* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *satisfies completeness. Moreover, our FDBE satisfies key extractability. Finally, assuming that: (i) the augmented* n*-bilinear Diffie-Hellman exponent assumption holds over the group* $\mathbb{G}$*, (ii)* $(\mathcal{E}, \mathcal{D})$ *is IND-CPA secure, and (iii)* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *satisfies knowledge soundness and zero-knowledge, our proposed FDBE scheme satisfies confidentiality.*

In the rest of this section, we prove the above theorem by stating and proving Theorem 4.3 (Section 4.4), Theorem 4.5 (Section 4.5), and Theorem 4.7 (Section 4.6), that establish correctness, key extractability, and confidentiality, respectively.

4.4 Analysis of Correctness

We state and prove the following theorem:

Theorem 4.3. *Assuming that the symmetric-key encryption scheme* $(\mathcal{E}, \mathcal{D})$ *satisfies correctness and* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *satisfies completeness, our proposed FDBE scheme satisfies correctness.*

To prove correctness of our proposed FDBE scheme, we first state and prove the following lemma.

Lemma 4.4. $\forall j \in [n]$*:* $P_j = P^{x^j}$*,* $T_j = Q^{x/x_j^j}$*,* $\mathsf{dk}_j = Q^{x^j}$ *and* $x = \prod_{k=1}^{n} x_k$*.*

Proof. We prove by induction that:

$$\mathsf{pp}_n = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n)$$
$$P_j = P^{x^j} \ ; \ T_j = Q^{x'_j} \tag{4.4}$$
$$x = \prod_{k=1}^{n} x_k \ ; \ \forall j \leq n : \ x'_j = x^j / x_j^j \tag{4.5}$$

In fact, for $i = 1$, we have:

$$\mathsf{pp}_1 = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n)$$
$$P_j = P^{x^j} \ ; \ T_j = Q^{x'_j}$$
$$x = x_1 \ ; \ x'_1 = x/x_1 = 1 \ ; \ \forall j \in [2, n] : x'_j = x^j$$

Now assume that for all $i < n$:

$$\mathsf{pp}_i = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n)$$
$$P_j = P^{x^j} \ ; \ T_j = Q^{x'_j}$$
$$x = \prod_{k=1}^{i} x_k \ ; \ \forall j \leq i : x'_j = x^j / x_j^j \ ; \ \forall j \in [i, n] : x'_j = x^j$$

This implies, in particular, that for $i = n - 1$:

$$\mathsf{pp}_{n-1} = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1 ..., T_n)$$
$$P_j = P^{x^j} \ ; \ T_j = Q^{x'_j}$$
$$x = \prod_{k=1}^{n-1} x_k \ ; \ \forall j < n : x'_j = x^j / x_j^j \ ; \ x'_n = x^n$$

Calling Update on input (pp_{n-1}, x_n) yields

$$\mathsf{pp}_n = (P, P_1^{x_n}, ..., P_n^{x_n}, P_{n+2}^{x_n^{n+2}}, ..., P_{2n}^{x_n^{2n}}, Q, T_1^{x_n}, ..., T_{n-1}^{x_n^{n-1}}, T_n)$$

and one can easily check that pp_n satisfies Equations 4.4 and 4.5. This entails that $T_j = Q^{x^j/x_j^j}$ and the honest party assigned broadcast slot j can correctly compute her decryption $\mathsf{dk}_j = T_j^{x_j^j} = Q^{x^j}$, where $x = \prod_{j=1}^{n} x_j$. □

Given Lemma 4.4, the correctness of our DBE is implied by the correctness of the symmetric encryption scheme $(\mathcal{E}, \mathcal{D})$ and completeness of $(\mathcal{G}, \mathcal{P}, \mathcal{V})$.

4.5 Analysis of Key Extractability

Theorem 4.5. *Our proposed FDBE scheme satisfies key extractability.*

Proof. Note that if $\mathsf{VerifyInit}$ accepts public parameters pp_0 provided by adversary $\mathcal{A}$, then this signifies that $\mathsf{pp}_0 = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n)$ where $P_j = P$ and $T_j = Q$.

Let i denote the index of the first time that $\mathcal{A}$ updates the public parameters and let (pp_i, π_i) denotes $\mathcal{A}$'s output. $\mathsf{VerifyUpdate}$ is provided then with input $(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i)$. We recall that:

$$\mathsf{pp}_{i-1} = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$$
$$\mathsf{pp}_i = (P, \{P'_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T'_j\}_{j\in[n]})$$

When $\mathsf{VerifyUpdate}$ accepts then we have:

$$j \notin \{n, n+1\} : e(P'_1, P'_j) = e(P, P'_{j+1}) \tag{4.6}$$
$$e(P'_2, P'_n) = e(P, P'_{n+2}) \tag{4.7}$$
$$j \neq i : e(T'_j, P_j) = e(T_j, P'_j) \tag{4.8}$$

Since this is the first time that $\mathcal{A}$ updates the public parameters, then we can write that, for some trapdoor x, $P_j = P^{x^j}$.

We recall that if $\mathcal{A}$ honestly executes Update, then $P'_j = P_j^{x_i^j}$ for $j \neq n+1$, $T'_j = T_j^{x_i^j}$ for $j \neq i$, and $T'_i = T_i$. We now show that if $\mathcal{A}$ did not execute Update as prescribed by the protocol, then the verification equations will not hold. We set $P'_1 = P_1^{x_i}$ for some x_i. Assume that the verification equations hold, but there is an index $l \in [2, 2n] \setminus n+1$ such that $P'_l \neq P_l^{x_i^l}$. Let k be the smallest index such that $P'_j \neq P_j^{x_i^j}$. Therefore, for all $j < k : P'_j = P_j^{x_i^j} = P^{(x_i x)^j}$. We now consider two cases: **Case** $k \neq n+2$**:** given Equation 4.6, we have $e(P'_1, P'_{k-1}) = e(P, P'_k)$ and this entails that $P'_k = P^{(x_i x)^k} = P_k^{x_i^k}$. **Case** $k = n+2$**:** Equation 4.7 implies that $P'_{n+2} = P^{(x_i x)^{n+2}} = P_{n+2}^{x_i^{n+2}}$.

In both cases, we contradict the fact that k is the smallest index such than $P'_j \neq P_j^{x_i^j}$.

Since we established that $\forall j \neq n+1$: $P'_j = P_j^{x_i^j}$, Equation 4.8 entails that $T'_j = T_j^{x_i^j}$ for $j \in [n]\setminus i$.

Hence, we conclude that if $\mathsf{VerifyUpdate}$ accepts an update by $\mathcal{A}$, then this implies that $\mathcal{A}$'s output was computed following the protocol. This also implies that honest parties will always be able to correctly compute their decryptions keys and decrypt the relevant ciphertexts. □

4.6 Analysis of Confidentiality

We first prove that the centralized variant of our scheme satisfies the confidentiality definition. In this variant of the scheme, we assume that there is a centralized setup that outputs the public parameters and supplies the parties with their decryption keys. We also note that in the centralized variant, the broadcast slot of party $\mathfrak{p}_i$ is i.

Theorem 4.6. *Assuming (i) the augmented n-BDHE assumption holds over $\mathbb{G}$ and (ii) $(\mathcal{E}, \mathcal{D})$ is a semantically-secure symmetric encryption scheme with key space $\mathcal{K}$, the centralized variant of our scheme satisfies confidentiality in the random oracle model.*

Proof. In the centralized setting, the confidentiality experiment does not have the adversary initialize or update the public parameters. Instead, right after receiving the challenge set of honest recipients, the challenger provides the public parameters and the decryption keys of the corrupt parties.

The proof proceeds via a sequence of hybrids:

- **Hybrid 0.** Identical to the confidentiality experiment for $b = 0$, i.e., the challenge ciphertext is a valid encryption of m_0.
- **Hybrid 1.** Identical to **Hybrid 1** except that the challenger sets $K = \mathcal{H}(\Omega')$ for uniformly-random $\Omega' \leftarrow \mathbb{G}_T$ in the challenge ciphertext.
- **Hybrid 2.** Identical to **Hybrid 2** except that the challenger sets $K \leftarrow \mathcal{K}$ in the challenge ciphertext.
- **Hybrid 3.** Identical to **Hybrid 3** except that the challenger sets $C = \mathcal{E}(K, m_1)$ in the challenge ciphertext.
- **Hybrid 4.** Identical to **Hybrid 4** except that the challenger sets $K = \mathcal{H}(\Omega')$ for uniform $\Omega' \leftarrow \mathbb{G}_T$ in the challenge ciphertext.
- **Hybrid 5.** Identical to the real broadcast encryption game for $b = 1$, i.e., the challenge ciphertext is a valid encryption of m_1.

In all of the hybrids, any query to random oracle $\mathcal{H}$ is answered using a uniformly randomly sampled bit string from $\{0,1\}^\kappa$. The proof of indistinguishability of the consecutive hybrids proceeds as follows.

Hybrid 0 $\approx_c$ Hybrid 1. We prove that Hybrid 0 is computationally indistinguishable from Hybrid 1 under the augmented n-BDHE assumption as follows.

The challenger receives as input a tuple of the form $\left(P, \{P_i\}_{i\in[1,2n]\setminus\{n+1\}}, \{T_{i,j}\}_{i\in[2n],j\in[n]}, H, \Gamma\right)$ where P and H are uniform random generators for $\mathbb{G}$, $P_i = P^{x^i}$, $T_{i,j} = P_i^{1/x_j^j}$, $x = \prod_{i=1}^n x_i$, and $\Gamma \in G_T$ is distributed as either $\Gamma = e(P,H)^{x^{n+1}}$ or $\Gamma \leftarrow \mathbb{G}_T$.

The adversary $\mathcal{A}$ outputs a challenge set $\mathcal{R} \subseteq [n]$ of honest parties (in other words, she corrupts each party $\mathfrak{p}_j$ for $j \notin \mathcal{R}$).

Now, when random oracle $\mathcal{H}_\mathbb{G}$ is called with inputs 0 and 1, the challenger responds with $\mathcal{H}_\mathbb{G}(0) = P$ and $\mathcal{H}_\mathbb{G}(1) = Q = P^a \left(\prod_{j\in\mathcal{R}} P_{n+1-j}\right)^{-1}$ fpr some random $a \leftarrow \mathbb{F}_p$. We recall that, in the centralized variant of the scheme, the broadcast slots of $\mathcal{R}$ coincide with $\mathcal{R}$.

Next, the challenger outputs the public parameters

$$\mathsf{pp}_n = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n)$$
$$\forall i \in [n] : T_i = T_{i,i}^a \left(\prod_{j \in \mathcal{R}} T_{n+1-j+i,i} \right)^{-1} .$$

We recall that $T_{i,j} = P_i^{1/x_j^j}$. Therefore, $T_i = Q^{x^i/x_i^i}$, and as a result, the public parameters are well-formed. Next, for each corrupt party $\mathfrak{p}_i$ the challenger provides decryption key

$$\mathsf{dk}_i = P_i^a \cdot \left(\prod_{j \in \mathcal{R}} P_{n+1-j+i} \right)^{-1} = Q^{x^i} = T_i^{x_i^i}.$$

Finally, The challenge ciphertext encrypting m_0 is created as $\mathsf{ct}^* = (h, C)$, where

$$h = (H, H^a) \; ; \; K = \mathcal{H}(\Gamma) \; ; \; C = \mathcal{E}(K, m_0)$$

Notice that if we write $H = P^r$, then $H^a = P^{ar} = (Q \prod_{j \in \mathcal{R}} P_{n+1-j})^r$. Therefore, when $\Gamma = e(P, H)^{x^{n+1}}$, the view of the adversary $\mathcal{A}$ is identical to that in Hybrid 0, and when $\Gamma \leftarrow \mathbb{G}_T$, the view of the adversary $\mathcal{A}$ is identical to that in Hybrid 1. Hence, any poly-time algorithm that distinguishes Hybrid 0 and Hybrid 1 can be used to create a poly-time attacker on the augmented n-BDHE assumption, leading to a contradiction.

Hybrid 1 $\equiv$ Hybrid 2. Hybrid 1 is identical to Hybrid 2 assuming that $\mathcal{H}$ is a random oracle since the distribution of K is uniform in both hybrids.

Hybrid 2 $\approx_c$ Hybrid 3. Hybrid 3 is computationally indistinguishable from Hybrid 2 assuming that $(\mathcal{E}, \mathcal{D})$ is a semantically-secure symmetric encryption scheme. Observe that in both Hybrid 2 and Hybrid 3, we have $K \leftarrow \mathcal{K}$, and the header component h in the challenge ciphertext $\mathsf{ct}^* = (h, C)$ does not depend on the plaintext message, and hence, can be simulated independently of the message. Therefore, when $C = \mathcal{E}(K, m_0)$, the view of the adversary $\mathcal{A}$ is identical to that in Hybrid 2, and when $C = \mathcal{E}(K, m_1)$, the view of the adversary $\mathcal{A}$ is identical to that in Hybrid 3. Thus, any poly-time algorithm that distinguishes hybrids 2 and 3 can be used to create a poly-time algorithm that, given m_0 and m_1, distinguishes $\mathcal{E}(K, m_0)$ and $\mathcal{E}(K, m_1)$ for uniformly random $K \leftarrow \mathcal{K}$. This breaks the semantic security of $(\mathcal{E}, \mathcal{D})$, leading to a contradiction.

Hybrid 3 $\equiv$ Hybrid 4. Hybrid 4 is again identical to Hybrid 3 assuming that $\mathcal{H}$ is a random oracle, since the distribution of K is uniform in both hybrids.

Hybrid 4 $\approx_c$ Hybrid 5. Finally, Hybrid 5 is computationally indistinguishable from Hybrid 4 under the n-BDHE assumption. The argument is analogous to that for the indistinguishability of Hybrid 0 and Hybrid 1 (we construct the challenger in exactly the same way except that the challenge ciphertext encrypts m_1 instead of m_0), and is hence not detailed.

This completes the proof of Theorem 4.6. □

Theorem 4.7. *Assuming (i) the augmented n-BDHE assumption holds over $\mathbb{G}$, (ii) $(\mathcal{E}, \mathcal{D})$ is a semantically-secure symmetric encryption scheme with key space $\mathcal{K}$ and (iii) and $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ satisfies knowledge-extraction soundness and zero-knowledge, our FDBE construction satisfies confidentiality in the random oracle model.*

Proof. Given the knowledge-extraction soundness and zero-knowledge properties of NIZK argument system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, we prove that if there is an adversary $\mathcal{A}$ who breaks the confidentiality of our scheme, then the challenger can leverage $\mathcal{A}$ to construct another adversary $\mathcal{B}$ that breaks the confidentiality of the centralized variant.

$\mathcal{A}$ first outputs the challenge set of honest parties $\mathcal{R}$. The challenger on behalf of the honest parties in $\mathcal{R}$ computes the broadcast slots Σ following the description of the protocol. If $n \notin \Sigma$, the challenger aborts. Otherwise, playing the role of adversary $\mathcal{B}$ against the confidentiality of the centralized scheme, the challenger outputs Σ. In return, the challenger receives the public parameters pp_n and the decryption keys dk_i for $i \notin \Sigma$.

$$\begin{aligned}
&\mathsf{pp}_n = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n) \\
&\forall i \in [2n] \setminus n+1 : P_i = P^{x^i} \ ; \ \forall i \in [n] : T_i = Q^{x^i / x_i^i} \\
&x = x_1 .... x_n \ ; \ \forall i \in \Sigma : \mathsf{dk}_i = Q^{x^i}
\end{aligned}$$

The challenger executes Update on behalf of all the honest parties with broadcast slots in $\Sigma \setminus \{n\}$. Let $x'_i, i \in \Sigma \setminus \{n\}$ denote the trapdoors that the challenger used. Moreover, when adversary $\mathcal{A}$ successfully updates the public parameters, the challenger leverages the knowledge extraction soundness of the proof π_i to extract the trapdoors x'_i used by $\mathcal{A}$. Now, let $\mathfrak{p}^*$ denote the honest party assigned broadcast slot n. The challenger simulates the Update call by $\mathfrak{p}^*$ as follows. Let $x'_1, ..., x'_{n-1}$ be the trapdoors that the challenger either extracted from the updates of $\mathcal{A}$ or used herself when calling Update, and let $x' = \prod_{i=1}^{n-1} x'_i$. The challenger simulates the computation of the public parameters by $\mathfrak{p}^*$ by outputting

$$\mathsf{pp}_n = (P, P_1, ..., P_n, P_{n+2}, ..., P_{2n}, Q, T_1, ..., T_n)$$

This is akin to the challenger using trapdoor $x'_n = x / \prod_{i=1}^{n-1} x'_i$ to update the public parameters.

What remains for the challenger is to compute a valid zero-knowledge proof of knowledge π_n. Accordingly, the challenger leverages the zero-knowledge property of zero-knowledge proofs to simulate π_n.

Next $\mathcal{A}$ outputs two messages m_0 and m_1, which the challenger relays as adversary $\mathcal{B}$. In response, the challenger receives ciphertext $\mathsf{ct}_b = (h, C)$, where

$$h = e\left(P^r, \left(Q \prod_{j \in \Sigma} P_{n+1-j}\right)^r\right);$$

$$K = \mathcal{H}(e(P_1, P_n)^r) \ ; \ C = \mathcal{E}(K, m_b).$$

She then provides ct_b to $\mathcal{A}$ and receives $\mathcal{A}$'s guess b^*. Finally, playing the role of adversary $\mathcal{B}$, the challenger returns bit b^*.

It follows that if adversary $\mathcal{A}$ has a non-negligible advantage ϵ in breaking the confidentiality of our FDBE, then adversary $\mathcal{B}$, and hence the challenger, will be able to break the confidentiality of the centralized scheme with non-negligible advantage $\geq \epsilon / n$. □

4.7 Additional Discussion

In this section, we present some additional discussion on our FDBE and the corresponding realization of Π_{CN}, called henceforth CN-FDBE.

Sequential Update Mechanism. CN-FDBE opts for sequential updates of the public parameters for two reasons. **(1)** If the parties concurrently update the public parameters, then they run the risk of seeing their updates rejected, as they may be updating stale public parameters (i.e., public parameters that have been updated between the time a party submits her Update transaction and the time the transaction is verified by the ledger). **(2)** The security of our FDBE's instantiation relies on a random oracle that assigns a broadcast slot to each one of the parties, and once that party is alloted a slot, she can only update the public parameters within the corresponding round, namely, the party with the last slot, updates the public parameters last, hence, mandating a sequential update.

Ledger and Network Assumptions. Our solution assumes that the consensus mechanism underlying the ledger is *censorship resistant*. This implies that Update transactions of honest parties will always be processed and included in the ledger within a bounded known delay Δ. We also assume that all the parties are online during their assigned rounds. Otherwise, an honest party may be unfairly excluded due to her not being able to submit her Update transaction in her round. We note that the above assumption (implicitly) requires the underlying network to be synchronous with a large Δ. We leave the study of relaxing this assumption as interesting future work.

Exclusion of Parties. If we define the duration of a round to be $\Delta_r = k \times \Delta$, then if an honest party submits her Update transaction before $(k-1) \times \Delta$ elapses, the chances that that transaction is not included in the ledger by the end of the round are slim (in fact, this could only happen due to connectivity issues). This follows from the censorship resistance property of the ledger. We, therefore, argue that if all the parties are online during their rounds and if Δ_r is defined appropriately, then the probability that an honest party is excluded from CN-FDBE is low. Finally, note that removal of a potentially honest party does not impact the security of other honest parties. In other words, the properties of confidentiality and key extractability would still hold with respect to the other honest parties.

On the off-chance that an honest party is excluded, CN-FDBE can be extended to support updates re-execution. More specifically, a party who missed her update round requests a freeze of the public parameters, and submits her Update transaction, out of round. Then, all the parties assigned the slots succeeding the slot of that party will be required to submit new Update transactions. In practice, that party submits a Freeze transaction that indicates to the smart contract that a re-execution should take place. Then, the smart contract verifies whether the origin of Freeze missed her round, waits until the current round elapses, and then starts processing Update transactions for slots $\geq i$, where i is the slot of the party requesting the freeze. This extension calls for the parties to be online until the phase of updating the public parameters finishes.

Handling Malicious Behavior. A malicious party can submit bogus Update transactions during her round. In this case, VerifyUpdate will fail and the transaction will be rejected. She can also submit either well-formed or bogus Update transactions outside her round. Also, in this case, VerifyUpdate will reject the transaction. A malicious party could also just not submit her Update transaction during her round, resutling in her exclusion. Now the malicious party must submit a Freeze transaction to be allowed back in, and she can abuse this feature to have CN-FDBE stuck in re-executions and not move forward. To prevent this scenario, we recommend a rate limiting

approach that restricts the number of times any given party can submit Freeze and successfully request re-execution.

4.8 Supporting Dynamic Addition of Parties

Our FDBE scheme can be naturally extended to support dynamic addition of parties as long as the maximum number of possible parties is upper bounded apriori by a fixed constant N. In this case, the original set of parties in the network initially execute a distributed setup with respect to the parameter N (note that only the public parameter size grows with N, and not the number of rounds required to set up the parameters, which is still equal to the initial network size). The only difference is that we do not use the lexicographic order after hashing the parties' identifiers to assign them the broadcast slots, instead we use universal hashing with range $[N]$.

Adding a New Party. Adding a new party to the network would proceed as follows. If the round assigned to the party has already passed, then the party is treated the same way as a party who missed her round in the original protocol. That is, a Freeze transaction is submitted, and a re-execution to produce the new public parameters is triggered. The original parties who already called Update and need to call it again, can reuse their trapdoor. If the round has not passed, then the party waits her round and updates the public parameters accordingly.

In both scenarios, the parties update the decryption keys to be consistent with the resulting public parameters. This simply requires each party to locally execute the key generation step using the updated public parameters and its own trapdoor (which does not change when the new party joins). Note that the correctness, security and efficiency guarantees follow in exactly the same way as the static version of the scheme.

Decrypting Historical Ciphertexts. We also note that the dynamic version of our FDBE scheme outlined above fully supports decrypting historical ciphertexts. Concretely, let N be the maximum number of possible parties. Informally, let "epoch"-j for any $j \in [N]$ denote the period between the timestamp when party $\mathfrak{p}_j$ joins and the timestamp when party $\mathfrak{p}_{j+1}$ joins. Let pp_j denote the set of public parameters associated with epoch-j, and let $\mathsf{dk}_{i,j}$ be the corresponding decryption key for any party $\mathfrak{p}_i$ for $i \in [j]$. Then, by the description of our FDBE scheme in Section 4.3, we have

$$\mathsf{dk}_{i,j} = \mathsf{ExtractDecKey}(pp_j, x_i)$$

where x_i is the trapdoor of party $\mathfrak{p}_i$, that remains unchanged across epochs. Hence, given any historical ciphertext ct associated with epoch-j such that $\mathfrak{p}_i$ for any $i \in [j]$ is in the intended list of recipients Σ of ct, $\mathfrak{p}_i$ simply does the following:

1. Look up pp_j (recorded publicly on the underlying ledger in an instantiation of Π_{CN} based on FDBE)
2. Derive the corresponding decryption key
$$\mathsf{dk}_{i,j} = \mathsf{ExtractDecKey}(pp_j, x_i)$$
3. Recover the plaintext message
$$m = \mathsf{Decrypt}(\mathsf{ct}, \Sigma, \mathsf{dk}_{i,j}, \mathsf{pp}_j)$$

Note that the above process only requires p_i to permanently store x_i (which matches our claim of constant-sized decryption keys) and the ledger to keep track of the updated public parameters. In particular, p_i need not permanently store $\mathsf{dk}_{i,j}$, since it can be computed on the fly as outlined above (given the public parameters of epoch-j).

Relaxing the Upper Bound. Note that one could relax the upper bound requirement by treating N as an epoch size instead, where each epoch allows a maximum of N insertions into the network, and the public parameter is reset at the end of each epoch. If N is set to be sufficiently large, this results in a reasonably infrequent resetting of the public parameters in practice.

5 Benchmarks and Application

5.1 Benchmarking Results for FDBE

In this subsection, we present benchmarking results for our FDBE scheme from Section 4.3. We also present a detailed comparison of FDBE with other pairing-based BE schemes with distributed key generation listed earlier in Table 1. We note that [CGPW25] and [CGPP24] essentially use the same underlying DBE scheme proposed in [GKPW24], so we just benchmark [GKPW24]. We use the BLS12-381 [BKLS02,Bow17] elliptic curve for the group and pairing operations, and AES-GCM for symmetric encryption/decryption. All results are reported on a VM running RHEL 9 with an 8-core CPU (each core running at 2.4GHz) and 32GB memory.

Component Sizes. In Table 3, we concretely compare FDBE with the other schemes from Table 1 in terms of the sizes of their public parameters, decryption keys, and ciphertexts. The reported figures follow the asymptotic analysis in Table 1. The size of public parameters for FDBE is significantly smaller than that of [WQZD10] and [KMW23]-2 (even though the latter requires a trusted setup), and marginally larger than that of [KMW23]-1 and [GKPW24]. The slightly larger public parameters is due to some additional pre-processed public parameter components, and is a tradeoff that allows FDBE to have significantly faster distributed key generation per-party as compared to [KMW23]-1 and [GKPW24] (illustrated subsequently). Finally, the decryption key and ciphertext sizes for FDBE are constant, and are either comparable to or smaller than that of the other schemes. Notably, [WQZD10] has non-constant key size (grows linearly with n) and $\approx 15\%$ larger ciphertext size than FDBE.

Key Generation. Table 4 compares the schemes from Table 1 in terms of the (public and decryption) key generation time per party. Key generation in FDBE is significantly more efficient than all other schemes and is effectively constant across different values of n (the small deviations are due to process variations across different runs). This is because key generation in FDBE involves just a single exponentiation. This is made possible by carefully pre-processing and including a few additional group elements in the public parameters of FDBE, as discussed earlier. In comparison, [KMW23]-1,2 and [GKPW24] require a linear (in n) number of exponentiations for key generation. The key generation time for [WQZD10] grows quadratically with n, and we could not even benchmark it for $n > 200$ as the process ran out of memory.

Encryption and Decryption. Fig. 6 compares FDBE with the other schemes from Table 1 in terms of the encryption and decryption overheads. Here, we present two sets of experiments where we fix the total number of parties to $n = 100$ and $n = 200$ (as mentioned earlier, the setup and key generation procedures for [WQZD10] ran out of memory for $n > 200$) and vary the target broadcast group sizes. We choose this setting because the encryption and decryption

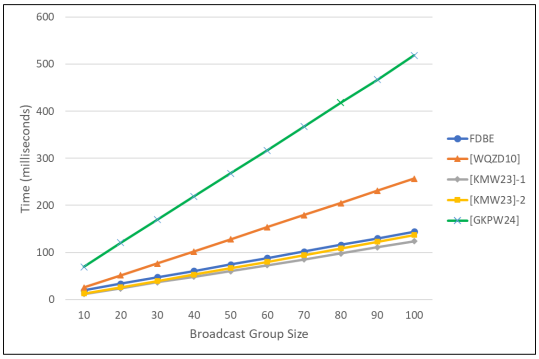
Scheme	Setup	n	$\|\mathsf{pp}\|$ (in KB)	$\|\mathsf{sk}\|$ (in KB)	$\|\mathsf{ct}\|$ (in KB)
[WQZD10]	Decentralized	10	17.27	2.30	0.85
		20	56.41	4.21	
		50	351.96	9.95	
		100	1157.03	19.53	
		200	4501.56	38.68	
		500	27660.15	96.05	
[KMW23]-1	Trusted	10	4.73	0.19	0.74
		20	9.65		
		50	24.42		
		100	49.0		
		200	98.25		
		500	245.91		
[KMW23]-2	Trusted	10	22.67	0.19	0.74
		20	83.09		
		50	494.96		
		100	1945.05		
		200	7710.67		
		500	48003.55		
[GKPW24]	Trusted	10	3.01	0.19	2.11
		20	6.02		
		50	15.04		
		100	30.08		
		200	60.16		
		500	150.39		
FDBE **(this work)**	Decentralized	10	4.29	0.19	0.74
		20	8.39		
		50	20.71		
		100	41.23		
		200	82.27		
		500	205.64		

Table 3: Comparison of concrete component-sizes. Here n denotes the total number of parties, $|\mathsf{pp}|$ denotes the size of the public parameters (including the public key), $|\mathsf{sk}|$ denotes the size of the decryption key for each party, and $|\mathsf{ct}|$ denotes the size of the ciphertext. [CGPW25, CGPP24, GKPW24] essentially use the same underlying DBE scheme proposed concretely in [GKPW24], so we just benchmark [GKPW24].

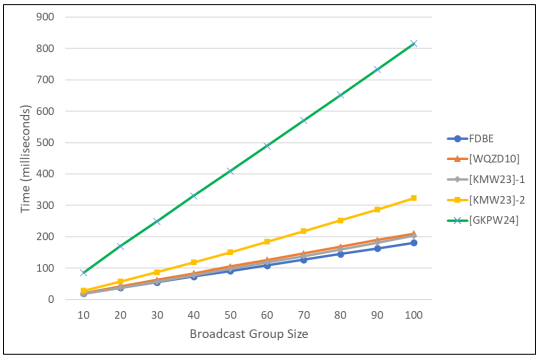
overheads for all of the schemes vary only with the target group size and not the total number of parties. The trends are similar for both $n = 100$ (figures (a) and (b)) and $n = 200$ (figures (c) and (d)). In particular, [GKPW24] performs significantly worse than all other schemes due to it highly

n	[WQZD10]	[KMW23]-1	[KMW23]-2	[GKPW24]	FDBE **(this work)**
10	273.60	42.60	89.40	44.51	1.17
20	889.00	86.65	185.90	89.74	1.15
50	4020.40	231.26	507.14	247.62	1.14
100	14231.71	506.24	992.58	512.49	1.12
200	58072.74	1002.76	1994.06	1031.56	1.16
500	–	2465.99	4953.86	2589.93	1.15

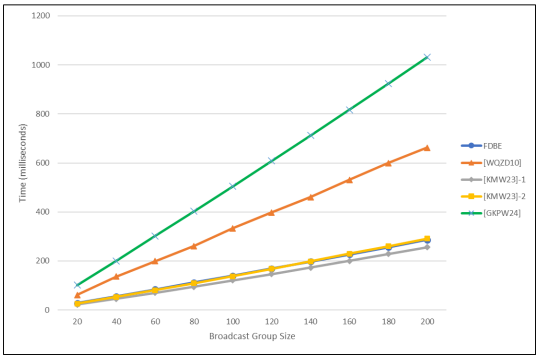
Table 4: Comparison of key generation time (in milliseconds) per party. Again, n denotes the total number of parties in the system. All measurements are averaged over 1000 experimental instances. We could not benchmark [WQZD10] for $n > 200$ as the process ran out of memory.



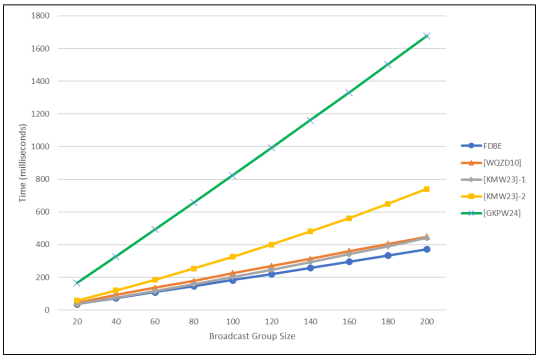
(a) Encryption Times ($n = 100$)



(b) Decryption Times ($n = 100$)



(c) Encryption Times ($n = 200$)



(d) Decryption Times ($n = 200$)

Figure 6: Comparison of encryption and decryption times (in milliseconds) for fixed n (the total number of parties in the system) and varying sizes of the target broadcast group. We present results for $n = 100$ and $n = 200$ (we could not benchmark [WQZD10] for $n > 200$ as the process ran out of memory, but expect the trends for other schemes to remain the same for $n > 200$). All measurements are averaged over 1000 experimental instances.

involved encryption and decryption algorithms. Overall, FDBE offers a desirable balance in terms of encryption and decryption performance. Notably, in comparison with [WQZD10] (which is the only other scheme with fully decentralized setup), FDBE supports *significantly faster* encryption *and* concretely faster decryption (this is because, unlike [WQZD10], FDBE avoids the heavy usage of target group operations for encryption and decryption).

5.2 Benchmarking Results for CN-FDBE

In this subsection, we practically demonstrate the applicability of CN-FDBE (the concrete instance of Π_{CN} based on the FDBE construction from Section 4.3) in enabling confidential interoperability between private blockchain/DLT networks. We present an augmentation of Hyperledger Cacti [Cac25], an open-source DLT interoperability framework, where networks built on Hyperledger Fabric [ABB+18] use CN-FDBE to share information confidentially (with integrity assurances additionally built in). Our implementation is available in the following fork of the Hyperledger Cacti GitHub repository: https://github.com/VRamakrishna/cacti/tree/crypto_dbe.

5.2.1 Hyperledger Cacti

Hyperledger Cacti is an open source project within Linux Foundation Decentralized Trust [LFD] that provides modules and libraries to enable transactions across networks leveraging similar or different DLTs, thereby interlinking their workflows. Cacti enables networks to communicate directly using centralized *node servers* or decentralized network-owned *relays*, hence, obviating reliance on third-party chains for communications and settlements [Cacb, Cacc]. Cacti supports protocols and provides reference implementations for networks to share (communicate) ledger data, atomically swap (exchange) and transfer assets (using community-agreed standards [HHSR23]), on demand [ABG+19]. Cacti provides DLT-independent communication *relays* and DLT-specific *connectors*, *drivers*, libraries, and smart contracts, for cross-network transactions and associated ledger state management, and supports networks built on Hyperledger Besu (an Ethereum client), Hyperledger Fabric (HLF), R3 Corda, ...etc.

Data Sharing in Cacti. Cacti offers a cross-network data sharing protocol; originally, this was created in Weaver, a separate interoperability framework, which is now part of Cacti [Cacg] (see Fig. 7(a)). Here, a smart contract in a *source network* receives a query (*view request* in Cacti parlance) for data (in its ledger) from an application (client) in a *destination network* (Steps 1-3). The client submits the ledger data (*view* in Cacti parlance) it receives to a smart contract in its network for validation, processing, and recording on its ledger (Step 9). For cross-network communication (Steps 1-2, 5-7), each network uses a *<relay, driver>* pair to open a *communication channel* with other networks. The source network's peers running the smart contract generate an authenticity proof (collectively, via the network's native consensus protocol) for the view data (Step 4). This proof is then independently validated by the destination network's peers (via that network's native consensus protocol). An authenticity proof typically consists of digital signatures (ECDSA for HLF). The communication channel consists of relays, drivers, and clients as *intermediaries* (see Section 1). The default mode of Cacti cross-network communication (called Cross-Network Plaintext or CN-PLAINTEXT henceforth) *provides no confidentiality* against the communication channel. In this mode, the *optional* operations in Fig. 7(a), i.e., encryption in the source network, decryption at the destination client, and matching the plaintext with the ciphertext during the validation step in destination network, are skipped.

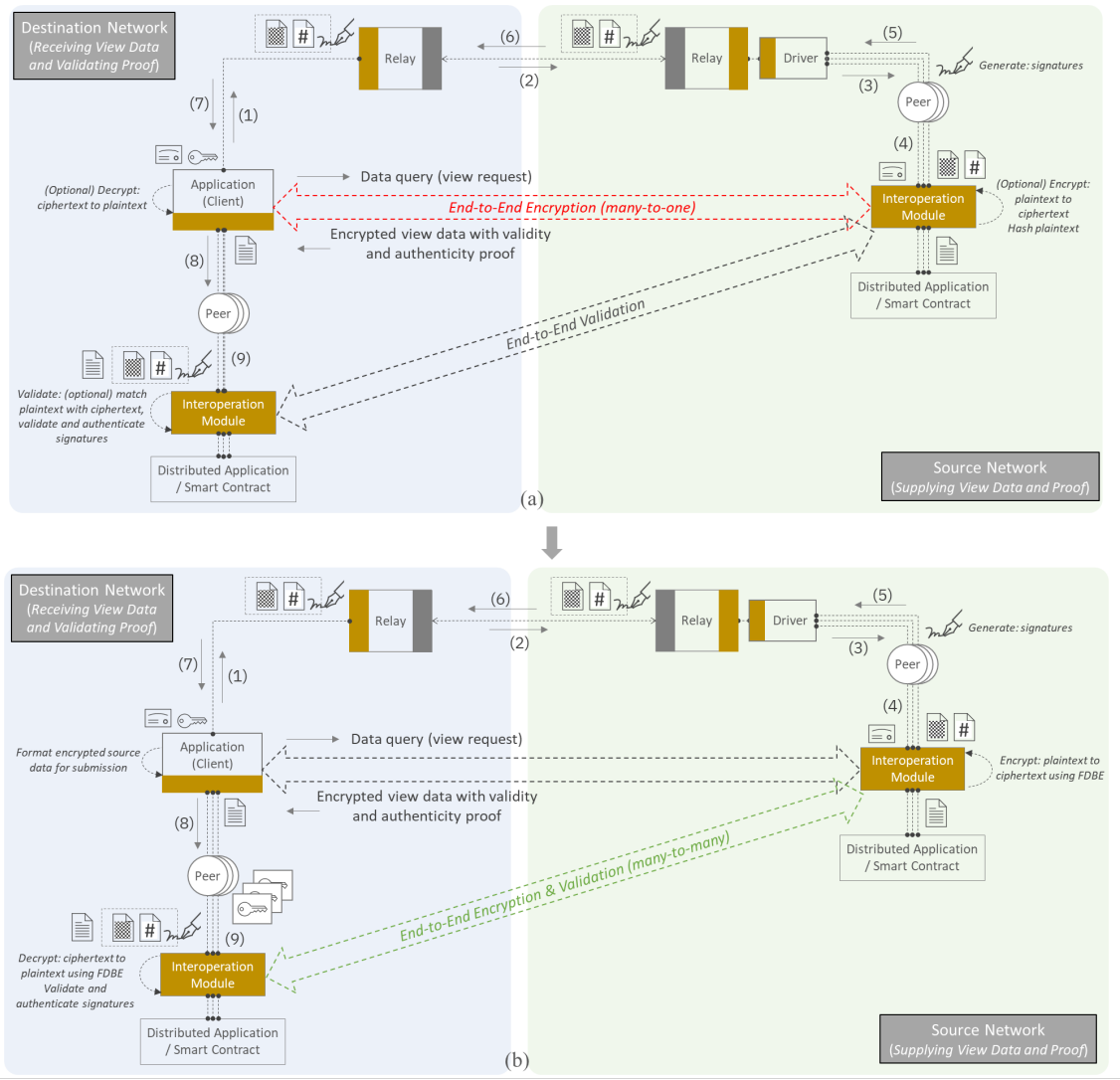


Figure 7: Cacti Data Sharing Protocol (images adapted from Hyperledger Cacti Weaver RFCs [Cacd]): (a) CN-PLAINTEXT when *optional* steps are skipped, and CN-PROXY when *optional* steps are performed; (b) Our implementation of CN-FDBE.

Confidential Data Sharing in Cacti. Cacti also offers a mode that adds confidentiality against a section of the communication channel, ensuring that malicious relays or drivers will not be privy to the plaintext view. This mode is called Cross-Network Proxy (or CN-PROXY for short), and it allows the client to decrypt the view, and essentially, act as a proxy for the peers of the destination network. After generating a view (with authenticity proof), the peers of the source network encrypt the view with the client's ECDSA public key, using ECIES. They also compute HMACs of the plaintext view (using pre-shared secret keys) so that the client cannot tamper with the plaintext view after

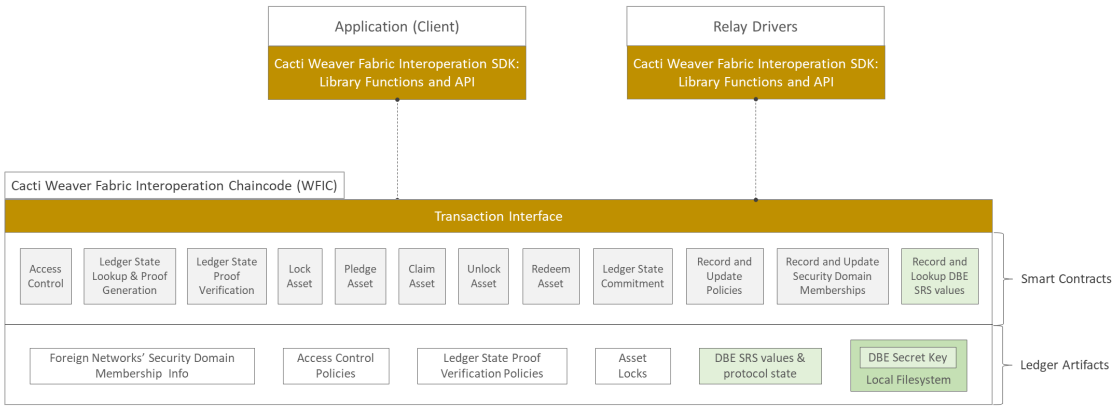


Figure 8: Cacti Weaver Fabric Interoperation Chaincode (WFIC) augmented with FDBE Capability

decryption and before validation by the peers of the destination network [Cacd]. However, as discussed in Section 1, this protocol inherently relies on a centralized trust assumption; indeed, *there is no confidentiality guarantee against a dishonest (potentially malicious) client/proxy, which is also part of the communication channel fro the perspectives of the source and destination networks' peer nodes.*

Our Implementation: CN-FDBE. We enhance Cacti with a new mode whereby the peers of the source network encrypt the view using FDBE (instead of ECIES) and the peers of the destination network decrypt the resulting ciphertext using the corresponding decryption keys. Unlike CN-PROXY, the client/proxy is no longer trusted to decrypt the data, and therefore, can no longer access the view before the peers of the destination network do. The view's authenticity proof is generated and validated exactly as in CN-PLAINTEXT and CN-PROXY. We label this mode Cross-Network FDBE or CN-FDBE (see Fig. 7(b)). While our implementation focused on HLF networks, the methods and APIs are portable to other DLTs (for e.g., Corda, Besu).

5.2.2 CN-FDBE – System and Implementation

Cacti offers logic for generic interoperation primitives (such as generation/verification of views and proofs and asset locks/unlocks) in *interoperation modules* [Cacf]. These are implemented as smart contracts for the supported DLTs, as their operations impact ledger integrity, and thus, must operate through consensus. In HLF, these interoperation modules are offered in the form of the *Weaver Fabric Interoperation Chaincode (WFIC)* (a *chaincode* is a smart contract in HLF parlance [Fabb]), which must be deployed on all the peers of an HLF ledger (also known as *channel*) to make it interoperable with other Cacti-enabled networks [Cace]. Fig. 7(a) illustrates *interoperation modules* in both networks.

Overview of the Implemented Functions. The following functions for CN-FDBE were implemented in the WFIC (in Go): (i) bootstrapping the ledger state in the destination network to initialize, sequentially update the public parameters, and finally, extract the decryption keys, one for each *organization* (in HLF, an organization is a sub-divison of the network's peers, acting as a redundancy set); (ii) encrypting data using FDBE in the source network, and (iii) decrypting

data using FDBE in the destination network. The Cacti client library for application clients, implemented as the *Cacti Weaver Fabric Interoperation Node SDK (WFIS)* (published as an NPM package [Caca]), required only one change for CN-FDBE; namely, skipping view decryption (required in CN-PROXY) and simply submitting it together with the authenticity proof to the destination's peers (this is identical to CN-PLAINTEXT). Moreover, no changes were required to the HLF relay and driver, which process and communicate messages exactly the same way as in CN-PLAINTEXT and CN-PROXY.

The WFIC, like any HLF chaincode, offers a transaction API that can be directly invoked by HLF clients, and which are backed by functions implemented within the chaincode. The WFIC architecture, enhanced with CN-FDBE capabilities, is illustrated in Fig. 8. Chaincode functions implemented for CN-FDBE setup are called out as "Record and Lookup DBE SRS Values" (SRS is "structured reference string, used to denote the public parameters in our FDBE scheme from Section 4.3). These functions support calls by the member organizations of the network to initialize and update the public parameters. Correspondingly, new ledger artifacts created for CN-FDBE are called out as "DBE SRS Values & Protocol State" and "DBE Secret Key (Local Filesystem)". (*Note*: the former is maintained in the shared ledger whereas the latter is maintained in each peer node's local file system). We also enhanced pre-existing chaincode functions – that handle pre-existing view requests and validations – with wrapper functions that generate and decrypt FDBE-encrypted payloads respectively. See Appendix A.1 for details of the modified WFIC.

Our implementation can be ported to other Cacti-supported DLTs (e.g., Corda, Besu) by implementing FDBE functions in the corresponding smart contracts and triggering functions in the corresponding client libraries.

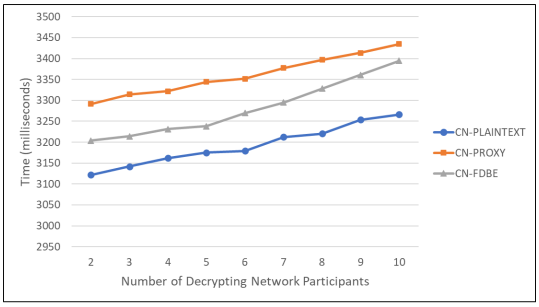
5.2.3 Cross-Fabric Network CN-FDBE Protocol

In this subsection, we present an abbreviated description of how our implementation of CN-FDBE is used to achieve confidential cross-network communication between two private HLF networks. Full details appear in Appendix A.2.

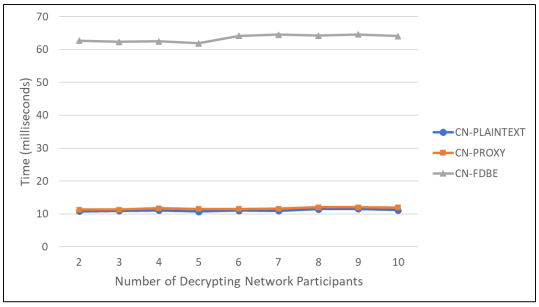
Bootstrap or Setup Phase. One or more organization members in the destination network trigger the initialization and validation public parameters. If the network has N organizations (parties), the public parameters update and corresponding validation are triggered N times in sequence, and each update triggered by a member of a different organization. Each update is idempotent (i.e., one organization may trigger an update exactly once) and its success depends on the previous steps having occurred in the right sequence, and the accepted/correct public parameters remain recorded on the ledger even if certain operations are duplicated or triggered out of order.

Data Sharing Protocol Instance. This consists of Steps 1 to 9 (Fig. 7(b)). Aside from the communication of view requests and responses, Step 4 allows the source network's peers to internally generate the requested view and calls the FDBE encryption function, while Step 9 internally calls FDBE decryption and validates the associated proofs in the destination network.

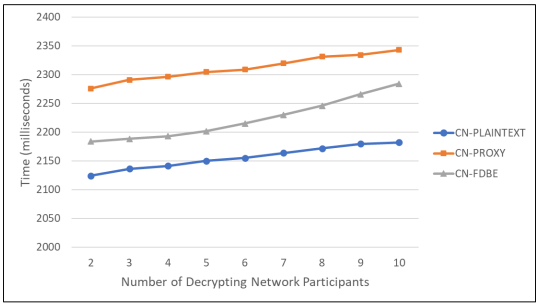
Update Phase. This occurs when a new organization joins a destination network with new peer nodes. It involves additional steps for updating and validating the public parameters for the latest (incremented) party count. The new peers are now ready to decrypt views encrypted using CN-FDBE.



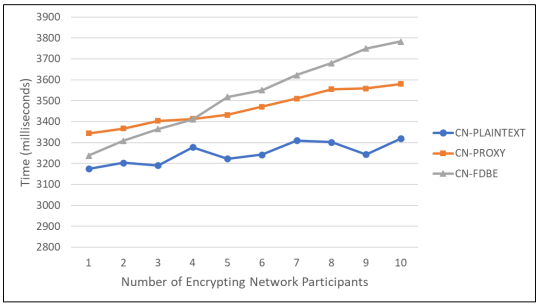
(a) End-to-End Latency for Single-Org Source Network



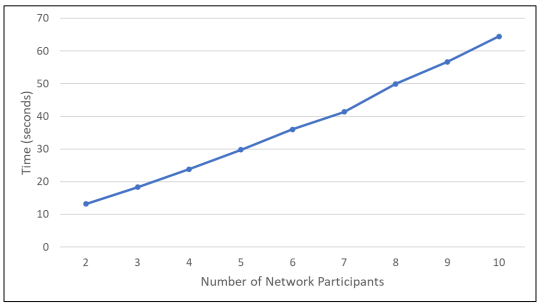
(b) View Generation Latency for Single-Org Source Network



(c) View Validation Latency for Single-Org Source Network



(d) End-to-End Latency for 5-Org Destination Network



(e) Total CN-FDBE Ledger Setup Time

Figure 9: CN-FDBE Performance: Measurements of Latency Across Two Networks (a-d) and Single-Network Setup (e)

5.2.4 Benchmarks and Performance Evaluation

We benchmarked our implementation of CN-FDBE and compared it to existing implementations of CN-PLAINTEXT and CN-PROXY (as illustrated in Fig. 7). In particular, we measured the end-to-end latency (Steps 1-9) as well as the sub-phases of the protocol: view generation (Steps 3-4) and view validation (Steps 7-8). Our objective in running these measurements was to determine whether the overhead incurred by CN-FDBE is practically tolerable for the strictly higher level of security it provides compared to a state-of-the-art confidentiality mechanism like CN-PROXY.

Note: We did not attempt to measure/compare network bandwidth usage or throughput for the three implementations. Such measures, though interesting from an application's (and interoperability) perspective, are orthogonal to our contribution, which assumes the necessity and practicality of cross-network data sharing (see Section 1) and instead tackles potential attacks on the communication protocol. Furthermore, as shown in Section 5.1, our FDBE scheme requires less storage compared to [WQZD10], which is the only other FDBE scheme with a fully decentralized setup; indeed, we offer $O(n)$ reduction in the size of the public parameters and the decryption key of each party. Plus, network throughput does not provide a meaningful comparison measure against [WQZD10], as throughput varies with ciphertext size, which is $O(1)$ for both our FDBE and [WQZD10] (with our scheme being only a constant-factor better concretely). Not needing to measure network resource usage meant that we did not require multiple physical or virtual machines for our blockchain network deployments, and instead could conduct our experiments (described below) on a single VM.

Setup. We augmented the sample test networks (testnets) offered by Cacti to run measurements for HLF network pairs of various combinations of sizes (see the 'weaver/tests/network-setups/fabric' folder in our code repository). Each testnet was launched with (i) a configurable number of organizations (parties), each containing a peer node, a CA node for identity issuance, and a Weaver identity syncing agent (*description of this is beyond the scope of this paper*), (ii) a Weaver relay, and (iii) a Weaver driver. The Cacti WFIC was deployed on every peer in a testnet. In each experiment, a pair of testnets was launched on a single VM running RHEL 9 with an 8-core CPU (each core running at 2.4GHz) and 32GB memory using `docker compose` with individual services running in separate Docker containers. We note that this is a standard framework for testing/evaluating applications built using HLF/Cacti.

Experiments. We ran experiments where the source network had a single peer to generate a view but the destination network size (i.e., number of organizations) varies from 2 to 10. Our network sizes did not exceed 10 because (i) HLF networks larger than that on a single VM consume a lot of resources and can be unstable, and (ii) almost all private blockchain and DLT networks in practice have only a handful of participants (fewer than 10), so running larger networks does not lead to practically useful insights. In particular, we highlight that we design experiments using a real blockchain software-stack and real interoperability modules, as opposed to standalone simulation (which could scale easily to larger networks but does not necessarily yield meaningful benchmarks). Our experiments target builders and administrators of private networks concerned about the overhead imposed on data-sharing protocols by encryption mechanisms.

As the Cacti data sharing protocol was designed to communicate view data between peer groups, we also ran experiments where the destination network size was fixed (at 5 *orgs*) while the source network size varied from 1 to 10 *orgs*. In a source network of size k, there are k encrypting nodes, so each destination network peer in CN-FDBE must decrypt k views before validating and processing the view data through consensus. This naturally increases the destination network load in CN-FDBE but not in the other two protocols as they either do not need to decrypt (CN-PLAINTEXT) or decrypt once in the application client (CN-PROXY).

Results. We ran each protocol 300 times for each combination of the above testnet sizes and computed the average. Fig. 9(a) shows the changes in the end-to-end latency of the three protocols. As expected, the latency for each protocol increases with destination network size, roughly in a linear fashion, though CN-FDBE seems to have a higher slope compared to the other two. This indicates that CN-FDBE may perform similarly or worse than CN-PROXY for large network sizes, but for practical network sizes (10 or fewer), the former significantly outperforms the latter. Thus,

beyond enabling a fully decentralized mechanism for confidential interoperability across private networks, CN-FDBE also provides a significant performance advantage over CN-PROXY.

During view generation, CN-FDBE has significant overhead as compared to CN-PROXY and CN-PLAINTEXT (both with almost identical performance); see Fig. 9(b). This is expected as the FDBE encryption mechanism is heavier than the ECIES generation with HMAC used in CN-PROXY. But for view validation (Fig. 9(c)), CN-PROXY performs significantly worse than CN-FDBE. Since view generation is in the order of tens of milliseconds and view validation in the order of 2 seconds, the latter dominates the former in the end-to-end latency comparison, hence showing that CN-FDBE is a significant improvement over CN-PROXY. Additionally, as illustrated in Fig. 9(d), when we fix the destination network size to 5 organizations and vary the size of source (encrypting) network instead, we see that CN-PROXY clearly gives performance advantages for larger network sizes (greater than 4). But CN-FDBE outperforms CN-PROXY at lower numbers of encrypting network peers, which are often encountered in practice.

Analysis. CN-FDBE requires less processing within the client of the destination network compared to CN-PROXY, which calls for an ECIES decryption algorithm. Within the destination network peers, view validation using FDBE is faster than HMAC and authenticity proof verifications when the source network only has a single encrypting peer. But when the number of encrypting peers increases (beyond 4), each destination network peer must decrypt and compare payloads from multiple source peers using FDBE (in CN-FDBE), which incurs higher latency than HMAC and authenticity proof verifications (in CN-PROXY).

Fig. 9(e) shows that the setup time increases linearly with network size (duration of an update round is 6 seconds). Given that this is a one-time operation, we believe that this latency is an acceptable overhead in practice.

Summary. In summary, CN-FDBE not only does it provide a qualitative benefit over the state-of-the-art by enabling a fully-decentralized mechanism for confidential cross-network interoperability, but also achieves performance gains for a large subset of practical configurations.

6 Conclusion

We introduced CN-FDBE – a novel, decentralized, practically efficient, and provably secure protocol for confidential communication across private blockchain/DLT networks. We modeled confidential cross-network communication as an ideal functionality $\mathcal{F}_{\mathsf{CN}}$ in the simplified UC framework, and presented a protocol Π_{CN} that provably realizes this functionality while resisting static corruptions based on a fully distributed notion of broadcast encryption (FDBE) without trusted setup. We realized CN-FDBE as a concrete instance of Π_{CN} based on a new FDBE scheme with constant-sized keys and ciphertexts from bilinear pairing groups. Reference implementations of FDBE and CN-FDBE in Hyperledger Cacti were used to demonstrate practically efficient and confidential information-sharing between private Hyperledger Fabric networks. We leave it as an interesting open question to explore the applicability of CN-FDBE in other blockchain networks and interoperability frameworks, as well as in a broader class of applications requiring private communication between decentralized groups of participants.

Ethics Considerations

In this paper, we proposed a new cryptographic mechanism that enables decentralized and confidential interoperability across private blockchain/DLT networks. We believe that it is justifiable to use Hyperledger Cacti and Hyperledger Fabric for our experiments given: (i) both Cacti and Fabric are *open source projects* within the Linux Foundation Decentralized Trust [LFD], (ii) the data sharing mechanisms in Cacti that our experimentation and benchmarking focus on are publicly documented [Cacd], (iii) both Cacti and Fabric have been extensively used for experimentation and benchmarking in several prior works [ABB+18, ABC+23, HHSR23, NRVN22], (iv) our findings do not pose any risks to individuals or organizations/entities currently using Cacti and/or Fabric, and (v) our findings do not reveal any new vulnerabilities that need to be disclosed. We further attest that the research team involved in this work considered the ethics of this research, and we believe that the research was done ethically. We plan to contribute our code base (https://github.com/VRamakrishna/cacti/tree/crypto_dbe) to the Hyperledger Cacti project, with the hope of encouraging the adoption of the proposed framework in real-world confidential interoperability scenarios.

References

[ABB+18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *EuroSys*, pages 30:1–30:15. ACM, 2018.

[ABC+23] Elli Androulaki, Marcus Brandenburger, Angelo De Caro, Kaoutar Elkhiyaoui, Liran Funaro, Alexandros Filios, Yacov Manevich, Senthilnathan Natarajan, and Manish Sethi. A Framework for Resilient, Transparent, High-throughput, Privacy-Enabled Central Bank Digital Currencies. *IACR Cryptol. ePrint Arch.*, page 1717, 2023.

[ABG+19] Ermyas Abebe, Dushyant Behl, Chander Govindarajan, Yining Hu, Dileban Karunamoorthy, Petr Novotný, Vinayaka Pandit, Venkatraman Ramakrishna, and Christian Vecchiola. Enabling Enterprise Blockchain Interoperability with Trusted Data Transfer (Industry Track). In *Middleware*, pages 29–35. ACM, 2019.

[AFG24] Diego F Aranha, Georgios Fotiadis, and Aurore Guillevic. A short-list of pairing-friendly curves resistant to the special tnfs algorithm at the 192-bit security level (hal-04666521v1). URL, 2024.

[AWY20] Shweta Agrawal, Daniel Wichs, and Shota Yamada. Optimal Broadcast Encryption from LWE and Pairings in the Standard Model. In *TCC 2020*, pages 149–178, 2020.

[BCD+14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. URL, 2014.

[bdf21] Banque de france demos ledger interoperability in cbdc trials. *Finextra Editorial*, 2021.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO 2001*, volume 2139, pages 213–229. Springer, 2001.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability. In *ACM CCS 2018*, pages 913–930. ACM, 2018.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *CRYPTO 2005*, pages 258–275, 2005.

[BIL+18] Kumar Bhaskaran, Peter Ilfrich, Dain Liffman, Christian Vecchiola, Praveen Jayachandran, Apurva Kumar, Fabian Lim, Karthik Nandakumar, Zhengquan Qin, Venkatraman Ramakrishna, Ernie G. S. Teo, and Chun Hui Suen. Double-blind consent-driven data sharing on blockchain. In *2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018*, pages 385–391. IEEE Computer Society, 2018.

[BKLS02] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO 2002*, volume 2442, pages 354–368. Springer, 2002.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography SAC 2005*, volume 3897, pages 319–331, 2005.

[Bow17] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction (Zcash blog, March 11 2017). URL, 2017.

[BSF+24] Rafael Belchior, Jan Süßenguth, Qi Feng, Thomas Hardjono, André Vasconcelos, and Miguel Correia. A brief history of blockchain interoperability. *Commun. ACM*, 67(10):62–69, 2024.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *IEEE S&P 2007*, pages 321–334, 2007.

[BW05] François Brezing and Alfred Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005.

[BW06] Dan Boneh and Brent Waters. A Fully Collusion Resistant Broadcast, Trace, and Revoke System. In *ACM CCS 2006*, page 211–220, 2006.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty Key Exchange, Efficient Traitor Tracing, and More from Indistinguishability Obfuscation. In *CRYPTO 2014*, pages 480–499, 2014.

[Caca] Hyperledger Cacti - Weaver Fabric SDK. URL.

[Cacb] Hyperledger Cacti: Architecture. URL.

[Cacc] Hyperledger Cacti: Project Scope. URL.

[Cacd] Hyperledger Cacti: Weaver: End-to-End Confidentiality. URL.

[Cace] Hyperledger Cacti: Weaver: Fabric Interoperability Contracts. URL.

[Cacf] Hyperledger Cacti: Weaver: Interoperation Modules. URL.

[Cacg] Introducing Hyperledger Cacti, a multi-faceted pluggable interoperability framework. URL.

[Cac25] Linux foundation decentralized trust - hyperledger cacti, 2025. Accessed on 09-Jan-2025.

[Can01] Ran Canetti. Universally Composable Security: a New Paradigm for Cryptographic Protocols. In *IEEE FOCS 2001*, pages 136–145, 2001.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A Simpler Variant of Universally Composable Security for Standard Multiparty Computation. In *CRYPTO 2015*, pages 3–22, 2015.

[CGPP24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool Privacy via Batched Threshold Encryption: Attacks and Defenses. In *USENIX Security 2024*, 2024.

[CGPW25] Arka Rai Choudhuri, Sanjam Garg, Guru-Vamsi Policharla, and Mingyuan Wang. Practical mempool privacy via one-time setup batched threshold encryption. In *USENIX Security 2025*, pages 3477–3495, 2025.

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved Dual System ABE in Prime-Order Groups via Predicate Encodings. In *EUROCRYPT 2015*, pages 595–624, 2015.

[CKKR19] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. iUC: Flexible Universal Composability Made Simple. In *ASIACRYPT 2019*, volume 11923, pages 191–221, 2019.

[CKS24] Michele Ciampi, Aggelos Kiayias, and Yu Shen. Universal Composable Transaction Serialization with Order Fairness. In *CRYPTO 2024*, volume 14921, pages 147–180, 2024.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *ACM STOC 1986*, pages 364–369, 1986.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[DF03] Yevgeniy Dodis and Nelly Fazio. Public Key Broadcast Encryption for Stateless Receivers. In *Digital Rights Management*, pages 61–80, 2003.

[dog] DOGETHEREUM: We can do this Dogether! URL.

[dtca] Dtcc press releases: Project ion platform. URL. (Last accessed: Aug 22, 2022).

[dtcb] Dtcc: Swift explores blockchain interoperability to remove friction from tokenized asset settlement. URL. (Last accessed: Jun 8, 2023).

[Faba] Hyperledger Fabric: Endorsement Policies. URL.

[Fabb] Hyperledger Fabric: Smart Contracts and Chaincode. URL.

[FN94] Amos Fiat and Moni Naor. Broadcast Encryption. In *CRYPTO' 93*, pages 480–491, 1994.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO '86*, volume 263, pages 186–194, 1986.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023*, volume 14084, pages 498–531. Springer, 2023.

[GKL24] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. *J. ACM*, 71(4):25:1–25:49, 2024.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold Encryption with Silent Setup. In *CRYPTO 2024*, volume 14926, pages 352–386, 2024.

[GKW18] Romain Gay, Lucas Kowalczyk, and Hoeteck Wee. Tight Adaptively Secure Broadcast Encryption with Short Ciphertexts and Keys. In *Security and Cryptography for Networks*, pages 123–139, 2018.

[GRR+21] Mike Graf, Daniel Rausch, Viktoria Ronge, Christoph Egger, Ralf Küsters, and Dominique Schröder. A Security Framework for Distributed Ledgers. In *ACM CCS 2021*, pages 1043–1064. ACM, 2021.

[gsb] Global shipping business network. URL. (Last accessed: Aug 6, 2025).

[GW09] Craig Gentry and Brent Waters. Adaptive Security in Broadcast Encryption Systems (with Short Ciphertexts). In *EUROCRYPT 2009*, pages 171–188, 2009.

[HHSR23] Thomas Hardjono, Martin Hargreaves, Ned Smith, and Venkatraman Ramakrishna. Secure Asset Transfer (SAT) Interoperability Architecture (Active Internet-Draft). URL, 2023.

[hql22] Hqlax, j.p. morgan, ownera and wematch demonstrate a cross-ledger repo. *HQLAx Announcements*, 2022. (Last accessed: April 22, 2025).

[HS02] Dani Halevy and Adi Shamir. The LSD Broadcast Encryption Scheme. In *CRYPTO 2002*, pages 47–60, 2002.

[hsb22] The interoperability of cbdcs across networks and currencies. *HSBC Report*, 2022. (Last accessed: April 22, 2025).

[HSV06] Florian Hess, Nigel P. Smart, and Frederik Vercauteren. The ate pairing on elliptic curves. In *EUROCRYPT 2006*, volume 4004, pages 346–359. Springer, 2006.

[htl20] Hashed time-locked contract transactions. *Bitcoin Wik*, 2020. (Last accessed: May 13, 2022).

[ibm] Ibm food trust. URL. (Last accessed: Nov 30, 2021).

[KB16] Jae Kwon and Ethan Buchman. Cosmos Whitepaper. URL, 2016.

[KLS16] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of Proofs of Work with Sublinear Complexity. In *Financial Cryptography and Data Security - FC 2016*, volume 9604, pages 61–78, 2016.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT 2023*, volume 14442, pages 407–441. Springer, 2023.

[KMZ20] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive Proofs of Proof-of-Work. In *Financial Cryptography and Data Security - FC 2020*, volume 12059, pages 505–522, 2020.

[KSS08] Ezekiel J. Kachisa, Edward F. Schaefer, and Michael Scott. Constructing brezing-weng pairing-friendly elliptic curves using elements in the cyclotomic field. In *Pairing-Based Cryptography - Pairing 2008*, volume 5209, pages 126–135. Springer, 2008.

[LFD] Linux Foundation Decentralized Trust. URL.

[mar] Marco Polo Network. URL.

[Mil04] Victor S. Miller. The weil pairing, and its efficient calculation. *J. Cryptol.*, 17(4):235–261, 2004.

[Mil12] Andrew Miller. The High-Value-Hash Highway. Bitcoin Forum post. URL, 2012.

[MKC22] Alex Murray, Dennie Kim, and Jordan Combs. The Promise of a Decentralized Internet: What is Web 3.0 and How Can Firms Prepare? *Business Horizons*, 05 2022.

[MNT01] Atsuko Miyaji, Masaki Nakabayashi, and Shunzou Takano. New explicit conditions of elliptic curve traces for fr-reduction. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 84(5):1234–1243, 2001.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[NPS20] Jonas Nick, Andrew Poelstra, and Gregory Sanders. Liquid: A Bitcoin Sidechain. *Liquid White Paper*, 2020.

[NRVN22] Krishnasuri Narayanam, Venkatraman Ramakrishna, Dhinakaran Vinayagamurthy, and Sandeep Nishad. Atomic Cross-chain Exchanges of Shared Assets. URL, 2022.

[PB17] Joseph Poon and Vitalik Buterin. Plasma: Scalable Autonomous Smart Contracts. *White Paper*, pages 1–47, 2017.

[PD15] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network. *Scalable Off-Chain Instant Payments*, pages 20–46, 2015.

[Pla] Meta Platforms. WhatsApp | Secure and Reliable Free Private Messaging and Calling.

[Sho00] Victor Shoup. Practical Threshold Signatures. In *EUROCRYPT 2000*, 2000.

[Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer, 2009.

[sin] Project ubin: Central bank digital money using distributed ledger technology. URL. (Last accessed: May 29, 2025).

[Tat74] John T Tate. The arithmetic of elliptic curves. *Inventiones mathematicae*, 23(3):179–206, 1974.

[tra] TradeLens. URL.

[TSB19] Jason Teutsch, Michael Straka, and Dan Boneh. Retrofitting a Two-Way Peg between Blockchains. *arXiv preprint arXiv:1908.03999*, 2019.

[TSH22] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain Scaling Using Rollups: A Comprehensive Survey. *IEEE Access*, 10:93039–93054, 2022.

[twp] A Next-generation Smart Contract and Decentralized Application Platform. URL.

[Ver10] Frederik Vercauteren. Optimal pairings. In *IEEE Transactions on Information Theory*, volume 56, pages 455–461, 2010.

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[Wei38] André Weil. Zur algebraischen theorie der algebraischen funktionen.(aus einem brief an h. hasse.). 1938.

[wet] we.trade. URL. (Last accessed: May 13, 2022).

[Woo16] Gavin Wood. Polkadot: Vision for a Heterogeneous Multi-Chain Framework. URL, 2016.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS 2010*, pages 741–743, 2010.

[XWJ23] Jie Xu, Cong Wang, and Xiaohua Jia. A Survey of Blockchain Consensus Protocols. *ACM Comput. Surv.*, 55(13s), 2023.

[Yak18] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13, 2018.

[ZABZ+19] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. Sok: Communication across distributed ledgers. Cryptology ePrint Archive, Paper 2019/1128, 2019.

[zkp21] Zkporter: A Breakthrough in L2 Scaling. URL, 2021.

A Application and System-Building: Additional Details

In this section, we present additional implementation-level details on CN-FDBE implementation described in Section 5. For more low-level details, please see our code, which is available in a fork of the Hyperledger Cacti GitHub repository: https://github.com/VRamakrishna/cacti/tree/crypto_dbe.

A.1 Cacti Weaver Chaincode Augmentation for CN-FDBE

We made the following categories of changes to the Weaver Fabric Interoperation Chaincode (WFIC) [3] for implementation-level details of these functions:

- **Chaincode Transactions for CN-FDBE Setup.** We implemented functions to record and validate initial and updated srs (structured reference string, used to denote the public parameters in our FDBE scheme from Section 4.3) values for the different organizations that the network comprises of (and which are represented by one or more redundant peer nodes). The srs generation functions (`Generate*`) require a single party to update ledger state whereas the srs validation functions (`Validate*`) require all network peers to update ledger state through consensus. We implemented the distributed setup and key generation through a sequential round-wise mechanism where each round looks up the previously recorded srs value. The FDBE secret key for each network participant (organization) must be kept private, and is hence recorded in the peer's local filesystem.
- **Internal Functions for CN-FDBE Operations.** The core FDBE functions and the associated NIZK system were implemented using elliptic curve operations over the widely used BLS12-381 [BN05] curve. We implemented the hash functions using SHA-256 and the symmetric-key encryption using AES-128 in the Galois counter mode (GCM). In the WFIC, we also implemented wrappers over these FDBE functions to generate FDBE-encrypted payloads and to decrypt such payloads, and exposed these wrappers to other chaincode functions.
- **Adaptations of Pre-existing Chaincode Transactions.** Where the source networks receive view requests and generate views in response, we added an option to generate FDBE-encrypted payloads with CN-FDBE. Where views passed by the application client to the destination network are processed and validated, we added an option to decrypt FDBE-encrypted payloads with CN-FDBE. We implemented the above functions in the following folder of our code repository: 'weaver/core/network/fabric-interop-cc/contracts/interop/'. To trigger WFIC transactions for CN-FDBE setup, we wrote wrapper functions and CLI scripts within the 'weaver/samples/fabric/fabric-cli' folder.

Detailed descriptions of the various functions implemented for CN-FDBE in the Cacti Weaver Fabric Interoperation Chaincode (WFIC) are given below. For a high-level illustration of how these functions augment pre-existing functions in the Cacti WFIC, please refer to Figure 8. We first list the functions exposed as external chaincode transactions. Internally, these functions call core FDBE functions for srs generation and update.

- `GenerateDbeInitVal(numOrgs, seed)`: This takes the number of participants in a network (in Fabric, these are *organizations*) as numOrgs and an arbitrary string as seed, generates

[3] URL: https://github.com/VRamakrishna/cacti/tree/crypto_dbe/weaver/core/network/fabric-interop-cc/contracts/interop

an initial srs and records it in serialized form on the ledger, marking it as unvalidated. This consensus rule (*endorsement policy* in Fabric parlance) [Faba] requires only one organization peer node to approve this transaction.

- `ValidateDbeInitVal()`: This looks up an *unvalidated* srs (if it exists) from the ledger, runs a validation check, and marks it as `validated` on the ledger if the check passes. The endorsement policy requires all organizations to approve this transaction.

- `GenerateDbeUpdateVal(entityId)`: This takes a serial number (i.e., version), creates an srs (i.e., $\mathsf{srs}_{entityId}$) corresponding to that by looking up the previous validated version of the srs (i.e., $\mathsf{srs}_{\mathsf{entityId}-1}$ recorded on the ledger. If entityId is 0, then $\mathsf{srs}_{\mathsf{entityId}}$ is created from the initial srs (recorded using `GenerateDbeInitVal`). $\mathsf{srs}_{\mathsf{entityId}}$ is then recorded on the ledger, but marked as *unvalidated*. The entityId is also recorded on the ledger. The secret key corresponding to this organization is also generated and recorded in a local file (not on ledger, as it cannot be made public.) The endorsement policy requires a single endorsement from a peer of an organization that has not recorded any of the SRS updates yet. If the peer of an organization that has already recorded an SRS update attempts to invoke this function, it will fail.

- `ValidateDbeUpdateVal()`: This looks up the latest entityId from the ledger, then looks up $\mathsf{srs}_{\mathsf{entityId}}$ from the ledger. If it is unvalidated, it runs a validation check and marks it as validated on the ledger if the check passes. The endorsement policy requires all organizations to approve this transaction.

- `GetDbeUpdatePublicParams()`: This looks up the latest entityId from the ledger corresponding to which the $\mathsf{srs}_{\mathsf{entityId}}$ recorded on the ledger is marked as unvalidated. it extracts the distributed public parameters from this srs and returns it in serialized form along with the version (i.e., entityId). The output can be used (or passed to another entity) as an encryption key. Since this is a query function and only reads the ledger, it does not need to pass network consensus.

Library functions not directly exposed as external chaincode transactions but which are called by functions exposed as transactions are listed below. Internally, these functions call core FDBE functions for key and ciphertext generation.

- `DistKeyGen(distributedPublicParameters, index, secretKey)`: This function takes the distributed public parameters corresponding to an srs, the index of an entity (i.e., order in which that entity recorded the srs on the ledger), and the secret key of that entity. It outputs a decryption key.

- `Encrypt(plaintext, targetArray, publicKey, publicParameters)`: This encrypts plaintext using publicKey and publicParameters, both of which can be extracted from the "encryption key" returned by `GetDbeUpdatePublicParams`. targetArray consists of a set of indices (i.e., order in which an entity recorded an srs) corresponding to the entities who will be able to subsequently decrypt the output ciphertext.

- `Decrypt(ciphertext, decryptionKey, targetArray, index, publicParameters)`: This decrypts ciphertext (encrypted using `Encrypt`) using the 'decryptionKey' which is obtained from `DistKeyGen` with a valid secret key and index corresponding to the entity calling this function. targetArray consists of the target (decryption) set indices as in the above function.

- `generateDBEPayload(maxIndex, srs, message)`: This is used by the WFIC in source networks to generate a DBE-encrypted payload from message (plaintext data) upon a request from a remote application (see Figure 7b). maxIndex is used to generate a target array for decryption $[1, \ldots,$ maxIndex$]$. Internally, it calls `Encrypt`.
- `decryptDBEPayload(index, ciphertext, srs, secretKey)`: This is used by the WFIC in destination networks, where individual organization peers decrypt the ciphertext received from a source network via an application (see Figure 7b) using their respective private keys stored locally on their file systems. Internally, it calls `Decrypt`.

Preexisting chaincode functions exposed as transactions for the data sharing protocol, modified to support CN-FDBE, are listed below.

- `HandleExternalRequest(...)`: This handles incoming requests from remote entities via relays. Internally, it performs access control and authentication checks before fetching or computing the requested data from the ledger and then packaging it. It can optionally encrypt this data before packaging, and we added an option to call `generateDBEPayload` so it was capable of performing CN-FDBE encryption.
- `WriteExternalState(...)`: This handles incoming data with associated proof from remote networks via relays and applications. Internally, it performs several validation and authentication checks before submitting the data to the business workflow it is needed in. We added a check to determine if the data was encrypted using CN-FDBE, and if so, inserted a call to `decryptDBEPayload`.

A.2 CN-FDBE Protocol

The complete protocol to prepare the respective networks and to run confidential data communications between them using CN-FDBE is described below. For an illustration, see Figure 7.

Bootstrap or Setup Phase. This only needs to be done on a network acting as destination, i.e., receiving and decrypting data. The transactions mentioned below are submitted to the WFIC deployed on the network.

- Submit `GenerateDbeInitVal` with a random seed and the number of organizations in the network (numOrgs) as arguments.
- Submit `ValidateDbeInitVal`.
- Initialize counter count $= 1$.
- For every organization in the network (count $= 1$ to numOrgs):
  - Submit `GenerateDbeUpdateVal` with count as argument. This transaction must be submitted by a client belonging to the given organization. If an organization has already submitted an update in an earlier round, this transaction will fail. A successful transaction will also create a secret key which only the organization's peers in the network will be privy to (in their respective filesystems).
  - Submit `ValidateDbeUpdateVal`.

- Now, peers belonging to any subset of the network's organizations are ready to decrypt CN-FDBE-encrypted data targeted for them.

Data Sharing Protocol Instance. This is as described in Figure 7b.

- Data request from an application in the destination network reaches source network's relay (Messages 1-2).
- Source network's relay driver submits `HandleExternalRequest` to the WFIC deployed on the source network (Message 3). Internally, this calls `generateDBEPayload]`, which in turn calls *Encrypt* as described earlier. The encrypted data along with validation proofs (consisting of ECDSA digital signatures) from different peers are collected using the consensus protocol (because the WFIC is a smart contract), packaged and set to the network's relay driver (Message 4).
- The data package reaches the application in the destination network via relays (Messages 5-6).
- The application submits `WriteExternalState` with this data package to the WFIC deployed in the destination network (Message 7). Internally, this calls `decryptDBEPayload`, which in turn calls `Decrypt` as described earlier. Despite the fact that different organization peers in the network use different secret keys, their outputs will match (as per the CN-FDBE mechanism) and hence consensus about ledger state change is achieved. After decryption, the proofs in the package are validated. The data is then processed according to the pre-deployed business workflow.

Update Phase. This occurs when a new organization joins the network with new peers; it replicates a portion of the bootstrap phase.

- Submit `GenerateDbeUpdateVal` with the latest counter value (e.g., if the earlier number of organizations was numOrgs, the counter will now be numOrgs + 1). A new secret key is generated by the peers of this new organization.
- Submit `ValidateDbeUpdateVal`.

Now, peers belonging to any subset of the expanded network's organizations are ready to decrypt CN-FDBE-encrypted data targeted for them.

As we can see in this procedure, the application (client) never gets to see the plaintext and therefore cannot exfiltrate it to unauthorized entities. Moreover, it cannot tamper with the data either as the data carries proofs in the form of digital signatures from source network peers.