Distributed Broadcast Encryption for Confidential Interoperability across Private Blockchains

Angelo De Caro
IBM Research Zürich
adc@zurich.ibm.com

Kaoutar Elkhiyaoui
IBM Research Zürich
kao@zurich.ibm.com

Sandeep Nishad
IBM Research India
sandeep.nishad1@ibm.com

Sikhar Patranabis
IBM Research India
sikhar.patranabis@ibm.com

Venkatraman Ramakrishna
IBM Research India
vramakr2@in.ibm.com

***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. The resulting construction is secure in the simplified universal composability (UC) 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.

I. Introduction

Blockchain and distributed ledger technology (DLT) networks were originally designed for public access and open participation, as evidenced by the popular Bitcoin [62] and Ethereum [76] networks, and more recently, by networks like Algorand [44] and Solana [79]. 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, interest in *private* or *permissioned* DLTs arose within enterprises and governmental institutions, which adapted decentralized consensus-based shared ledgers for use as shared systems-of-record for transactions among members of private groups (or business consortiums) [27], [28]. Private DLTs, in combinations with public ones, have enabled applications like supply chains [5], [18], [13], trade finance [16], [19], central bank digital currency (CBDC) [17], [21], securities trading [3], [21], and regulatory compliance [33] to be built at scale. In these scenarios, mutually untrusting parties (like exporters, importers, and banks) cannot rely on common trusted central authorities to manage ledgers, yet need a trustworthy decentralized transaction processing method with audit trails for dispute resolutions.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.241200
www.ndss-symposium.org

Background on DLT Interoperability. DLT networks with different trust and governance models have proliferated in both public and private forms. This in turn has driven research in interoperability, a necessity for these networks to fulfill 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 [21], [24], [23]. Technically, interoperation enables transactions to span multiple networks, primarily to (i) share state and drive operations [26], (ii) exchange assets atomically [20], [63] (e.g., delivery-vs-payment (DvP) [21], [24], [23]), and (iii) transfer assets securely [53], [4]. All these patterns at their core rely on foolproof cross-network communication [80], 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 [61].

Communication solutions invariably involve *intermediary* components like bridges [32], settlement chains [75], [58], relay nodes [26], and even global messaging systems like SWIFT [4]. Some of these components are external to the intercommunicating networks whereas others lie at least partly within them (i.e., have partial access to network resources). Though designed for both public and private DLT networks, such intermediaries pose unique challenges and threats to the latter. This is because a private network is not simply a

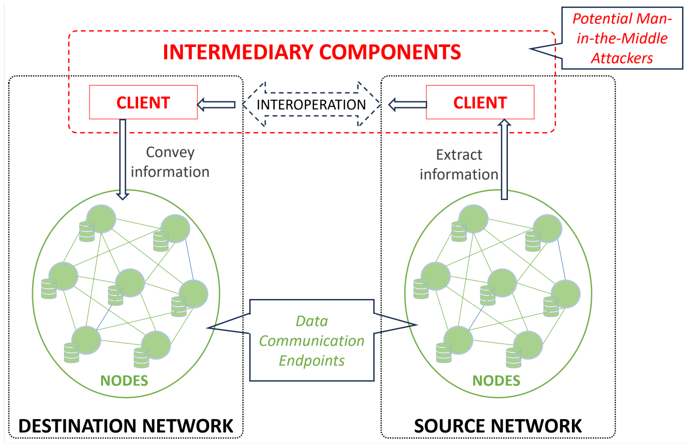


Fig. 1: Cross-Network Communication Model for Private DLTs

constrained form of a public network but is a distinct class of distributed systems. The ledger state and transaction logs of private networks, unlike those of public networks, are by default not visible to nor verifiable from the external world. But external visibility and verifiability are (by definition) *key requirements* for cross-network communication, without which transactions across DLT networks cannot be supported.

Communication intermediaries fulfill these requirements but introduce other threats, namely the ability to tamper with data integrity (relevant to both public and private networks) and to exfiltrate a network's ledger data (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. The threat model covers network nodes tampering with ledger and transaction integrity but does not cover private ledger state leakage to external entities. New threat vectors are introduced when private networks must communicate ledger data back-and-forth via intermediaries, which cannot be trusted to maintain the privacy of communicated data, unlike the network's nodes (members), which have a vested interest in keeping the data confidential among themselves. Prior interoperability research has focused on ensuring end-to-end communication integrity (*information possessed by nodes in one network must reach relevant nodes of another network without intermediaries being capable of tampering with the information*) 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 [1], [29], [2], [71], [59], [55], [56]. 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 [67], [66], [64], [22], [72]. To summarize, ensuring the integrity of communication across DLT networks/chains (independent 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 challenge of providing confidentiality (to some extent) in addition to integrity is the data sharing protocol provided by Hyperledger Cacti [25], [9], but 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 decouple 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 [78] and threshold signatures [68]. 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 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 [65] 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 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 ([25], [9]) 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 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?*

A. 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 simplified universal composability (sUC) framework from [42]. As noted in [42], the sUC framework is a practically meaningful, easy-to-use alternative to the traditional UC framework from [41]. For simplicity of exposition, our functionality uses a simplified abstraction of a private DLT network (and the corresponding private ledger), and focuses on capturing the core data confidentiality requirements. The detailed description appears in Section III.

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) [49], [48], [36], [38], [39], [52], [57], [51], [45], [46]. We call this primitive fully distributed BE (or FDBE in short). We formally define FDBE in Section IV-A. We then describe our FDBE-based protocol Π_{CN} and prove its security in the sUC framework in Section IV-B.

Scheme	Setup	$\|\mathsf{pp}\|$	$\|\mathsf{sk}\|$	$\|\mathsf{ct}\|$
[77]	Decentralized	$O(n^2)$	$O(n)$	$O(1)$
[57]-1	Trusted	$O(n)$	$O(1)$	$O(1)$
[57]-2	Trusted	$O(n^2)$	$O(1)$	$O(1)$
[46], [45], [51]	Trusted	$O(n)$	$O(1)$	$O(1)$
FDBE (this work)	Decentralized	$O(n)$	$O(1)$	$O(1)$

TABLE I: 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. [46] and [45] essentially use the same underlying DBE scheme proposed in [51], so we group them together.

Concrete Instantiation of FDBE. We present the first FDBE scheme with constant-sized keys and ciphertexts that (i) supports fully decentralized setup and distributed key generation, (ii) achieves comparable encryption and decryption efficiency to the most efficient, trusted-setup based BE schemes (such as [36], [38], [52]), (iii) relies on cryptographic hardness assumptions over bilinear groups, and (iv) is suitable for deployment in private blockchain/DLT networks (we expand more on this in the discussion below). Plugging this into our generic construction Π_{CN} yields an instantiation of confidential and decentralized cross-network communication from the same assumption with low bandwidth requirements and small key management overheads. We call this solution CN-FDBE. In particular, the encryption and decryption overheads of CN-FDBE 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 IV-C for the detailed construction of our FDBE scheme.

In Table I, we compare our FDBE scheme to recently proposed pairing-based BE constructions with distributed key generation [57], [51], [45], [46] (we note that [46] and [45] essentially use the same underlying DBE scheme proposed in [51]). 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 [77]. However, [77] 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 like [36] using techniques such as secure multiparty 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 [36] 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 [57], [51], [45], [46] 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 (as compared to stronger security guarantees) when a majority of the parties are maliciously corrupt [47]. With abort security, 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 its performance against the pairing-based BE schemes from Table I in Section V-A. We further demonstrate the practicality of the FDBE-based instantiation of Π_{CN}, which we call Cross-Network-FDBE (abbreviated as CN-FDBE henceforth), 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 V-B for details.

II. Preliminaries

This section presents preliminary background material.

A. Bilinear Pairings

We present background material on bilinear pairings, which are used crucially in our FDBE construction in Section IV-C.

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, while 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 [74], [60] or the Tate pairing [70], [69]. 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:*

- ***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, then $\exists Q \in \mathbb{G}_2 : e(P, Q) \neq 1$. Similarly, if $Q \in \mathbb{G}_2$ is a non-identity element, 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 (there exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$, but not conversely) and 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. More specifically, $\mathbb{G}_1$ is realized as an order-p additive subgroup of $E(\mathbb{F}_q)$ (same as in symmetric bilinear pairings), $\mathbb{G}_2$ is often realized as an order-p additive subgroup of $E(\mathbb{F}_{q^k})$ (k being the embedding degree), and $\mathbb{G}_T$ is often realized as an order-p multiplicative subgroup of $\mathbb{F}_{q^k}^\times$.

In practice, Type-3 pairings such as (optimal) Ate pairing [54], [73] 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 [30] and the Barreto–Naehrig (BN) family of elliptic curves [31].

Remark. We remark here that symmetric bilinear pairings are often used to describe cryptographic applications for simplicity purposes (see [34], [35], [37], [36], [38] for a non-exhaustive list of examples). We use the same approach in Section IV-C to describe the construction of FDBE. However, in practice, asymmetric bilinear pairings allow for more efficient instantiations. Accordingly, our implementation of FDBE also uses asymmetric bilinear pairings.

B. NIZK Arguments of Knowledge

We recall the formal definition of non-interactive zero-knowledge (NIZK) arguments of knowledge.

Definition 3 (Ternary Relation)**.** *A ternary relation $\mathcal{R}$ is defined by a triple $(\mathsf{pp}, \mathbb{x}, \omega)$ where pp is the public parameters, $\mathbb{x}$ the instance, and ω the witness. If triple $(\mathsf{pp}, \mathbb{x}, \omega)$ satisfies $\mathcal{R}$, then we write $\mathcal{R}(\mathsf{pp}, \mathbb{x}, \omega) = 1$. Else $\mathcal{R}(\mathsf{pp}, \mathbb{x}, \omega) = 0$. We refer to $\mathcal{L}_\mathcal{R} = \{(\mathsf{pp}, \mathbb{x}) : \exists \omega \text{ s.t. } \mathcal{R}(\mathsf{pp}, \mathbb{x}, \omega) = 1\}$ as the language of relation $\mathcal{R}$.*

Definition 4 (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 ω, 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* $\mathsf{tr} \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \omega), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle$.

The interaction between $\mathcal{P}$ and $\mathcal{V}$ concludes by having $\mathcal{V}$ output a bit $b = \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \omega), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle$. $b = 1$ 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 probabilistic polynomial time (PPT) adversaries $\mathcal{A}$:*

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

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:*

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

Definition 5 (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 $\mathsf{tr} \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \omega), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle$ leaks zero information about the witness ω. More formally:

Definition 6 (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:*

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

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

NIZK Arguments of Knowledge. In the random oracle model, public-coin interactive (zero-knowledge) arguments of knowledge can be made non-interactive using the Fiat-Shamir heuristic [50]. 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. $\mathcal{P}(\mathsf{pp}, \mathbb{x}, \omega)$ 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.

III. Modeling Confidential Cross-Network Communication in the Simplified UC Framework

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 [42].

A. 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 the consensus protocols underlying $\mathcal{N}_0$ and $\mathcal{N}_1$. Namely, if f_0 and f_1 are the corruption thresholds beyond which the security of the consensus protocols underlying $\mathcal{N}_0$ and $\mathcal{N}_1$ cannot be assured, then we assume that only f_0 (within $\mathcal{P}_0$) and f_1 (within $\mathcal{P}_1$) parties can be corrupted. For simplicity, we consider that the smart contract deployment and execution are out of scope. 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* $\mathsf{RemoteRead}$ 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 relay $\mathsf{RemoteRead}$ 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.

B. 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 $\mathsf{RemoteRead}$ 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 $\mathsf{RemoteRead}$ 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 $\mathsf{RemoteRead}$ 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 (i.e., *correctness*, *confidentiality*, and *authenticity*), we turn to the sUC framework [42], 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 $\mathcal{A}$ corrupt 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 $\mathcal{N}_1$ respectively); 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 parameterized with networks $\mathcal{N}_0$ and $\mathcal{N}_1$. Each network exposes a $\mathsf{CheckAuth}$ 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 $\mathsf{RemoteRead}$ and $\mathsf{SendValue}$.

$\mathsf{RemoteRead}$ *Interface.* $\mathsf{RemoteRead}$ 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 $\mathsf{RemoteRead}$ 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 $\mathsf{RemoteRead}$ 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 $\mathsf{RemoteRead}$ invocation, $\mathcal{S}$ learns all the information contained in the request.

$\mathsf{SendValue}$ *Interface.* $\mathsf{SendValue}$ 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 $\mathsf{RemoteRead}$ request rid. If $\mathfrak{p}$ is honest, then $\mathcal{F}_{\mathsf{CN}}$ sends to $\mathcal{N}_b$ a $\mathsf{CheckAuth}$ 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,

<u>RemoteRead</u> 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 $\mathcal{P}_b$.

<u>SendValue</u> 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 $\mathcal{S}$.

Fig. 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.

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 $\mathsf{RemoteRead}$ 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 $\mathsf{RemoteRead}$ 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 $\mathsf{SendValue}$ 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 honest 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 7 (Secure Cross-Network Communication)**.** *A cross-network communication protocol* Π *securely realizes* $\mathcal{F}_{\mathsf{CN}}$*, if for any PPT adversary* $\mathcal{A}$ *and any PPT environment* $\mathcal{Z}$*, there exists a PPT simulator* $\mathcal{S}$ *such that* $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \approx_c \text{IDEAL}_{\mathcal{F}_{\mathsf{CN}},\mathcal{S},\mathcal{Z}}$*, where* $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ *is the random variable denoting the output of* $\mathcal{Z}$ *in the real world,* $\text{IDEAL}_{\mathcal{F}_{\mathsf{CN}},\mathcal{S},\mathcal{Z}}$ *is the random variable denoting the output of* $\mathcal{Z}$ *in the ideal world, and* $\approx_c$ *denotes computational indistinguishability.*

IV. 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).

A. *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 party $\mathfrak{p}_i$ 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, $\mathsf{VerifyInit}$ 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, $\mathsf{VerifyUpdate}$ 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 $\mathfrak{p} \in \mathcal{P}$, $\mathsf{ExtractDecKey}$ 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, $\mathsf{Encrypt}$ computes ciphertext ct.

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

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 $\mathsf{out} \neq m$.

Fig. 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** $b = b^*$.

Fig. 4: Confidentiality Experiment for FDBE

Correctness and Security Definitions. We consider a static adversary $\mathcal{A}$ that corrupts parties before the assignment of broadcast slots and the calling of 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 yields public parameters that will always be accepted by VerifyInit, whereas subsequent honest executions of Update will result in public parameters that will always be accepted by VerifyUpdate. More formally, the following equalities hold:

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

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:

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

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.

B. 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, every party in $\mathcal{P}_0$ and $\mathcal{P}_1$ is endowed 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) ensures 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 $\mathcal{L}_b[\mathsf{FDBE.PP}] \rightarrow \mathsf{pp}_{b,i-1}$, 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 value $\mathsf{v} \leftarrow \mathcal{L}_{\bar{b}}[\mathsf{K}]$ from her local copy of the ledger; (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 IV.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 $\mathcal{F}_{\mathsf{CN}}$.*

We defer a detailed proof of this theorem to the full version of the paper [43].

C. Bilinear Pairing-based Construction of FDBE

In this section, we present an 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 IV-B. The starting point of our design is the bilinear pairing-based BE scheme with constant-size decryption keys and constant-size ciphertexts from [36]. The original scheme from [36] 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 and modify 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 [57], [51], [45], [46] that require trusted setup to generate a common reference string, our scheme completely 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 an FDBE scheme. Let $\mathbb{G}$ be a group of prime order p that admits a non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and let P and Q be two uniformly-random generators for $\mathbb{G}$ (see Section II-A for background material on pairings). Let $[n]$ denote the set of integers $\{1, ..., n\}$. 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 $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ and $\mathcal{H}_{\mathbb{G}} : \{0,1\}^* \rightarrow \mathbb{G}$ be cryptographic hash functions.

Building Blocks. We use the following primitives as building blocks: (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 soundness and zero knowledge (in the random oracle model, see Section II-B for the formal definition). In particular, the NIZK argument of knowledge is used to generate proofs for instance-witness pairs of the form $(\mathbb{x}, \omega) = ((P, P'), x)$ where $P, P' \in \mathbb{G}$ and $x \in \mathbb{F}_p$ satisfying the discrete log relation:

$$P' = P^x \tag{IV.1}$$

The Construction. We now present the detailed construction.

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 of 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 generates

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

Finally, it outputs pp_0. Observe that the generation of pp_0 does not involve any secrets.

$\mathsf{VerifyInit}(\mathsf{pp}_0, n) \to 0/1$: On input of

$$\mathsf{pp}_0 = (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$$

$\mathsf{VerifyInit}$ 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 $j \in [n]$[1]. Otherwise, it outputs 0.

Public Parameters Updates

$\mathsf{Update}(\mathsf{pp}_{i-1}, x) \to (\mathsf{pp}_i, \pi_i)$: Given pp_{i-1} and trapdoor $x \in \mathbb{F}_p$ from party assigned slot i, Update first parses $\mathsf{pp}_{i-1} := (P, \{P_j\}_{j\in[2n]\setminus\{n+1\}}, Q, \{T_j\}_{j\in[n]})$ and computes

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

Update also produces a NIZK proof $\pi_i \leftarrow \mathcal{P}((P'_1, P_1), x)$ for the relation in Eq. IV.1. Finally, Update outputs (pp_i, π_i).

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

$$j \notin \{n, n+1\} : e(P'_1, P'_j) = e(P, P'_{j+1}) \tag{IV.2}$$
$$e(P'_2, P'_n) = e(P, P'_{n+2}) \tag{IV.3}$$
$$j \neq i : e(T'_j, P_j) = e(T_j, P'_j)$$

If any of these checks fails, then $\mathsf{VerifyUpdate}$ rejects and outputs 0; otherwise, it accepts and outputs 1. Notice that thanks to the Schwartz-Zippel lemma, the verification equations in IV.2 and equation IV.3 can be aggregated into one equation. Actually, if $\mathsf{VerifyUpdate}$ randomly chooses $(\alpha, \beta) \in \mathbb{F}_p^2$ and the following equality holds:

$$e(P'_1, \prod_{j=1}^{n-1} P_j'^{\alpha^j} \prod_{j=n+2}^{2n-1} P_j'^{\alpha^j}) e(P_2'^{\beta}, P'_n) =$$
$$e(P, P'^{\beta}_{n+2} \prod_{j=1}^{n-1} P'^{\alpha^j}_{j+1} \prod_{j=n+2}^{2n-1} P'^{\alpha^j}_{j+1})$$

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

then $\mathsf{VerifyUpdate}$ can conclude that equations IV.2 and IV.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 a party in $\mathcal{P}$ misses her round of update, then the protocol moves onto the next round, and that party 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

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

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

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

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

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

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

Correctness and Security. Correctness follows immediately from the completeness of $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and correctness of $(\mathcal{E}, \mathcal{D})$, 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 [36].

Definition 8 (Augmented n-BDHE Assumption)**.** *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 H be uniformly random generators for $\mathbb{G}$. Let $x_1, ..., x_n \leftarrow \mathbb{F}_p$, $x = \prod_{i=1}^n x_i$, and*

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

For any PPT algorithm $\mathcal{A}$, and any bit $b \in \{0,1\}$, define

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

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 IV.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.*

Due to space limitations, we defer a detailed proof of this theorem to the full version of the paper [43].

D. Additional Discussion

In this section, we present some additional discussion on our FDBE and the corresponding realization of Π_{CN}, called hereafter 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; in particular, the party with the last slot must update 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. 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 re-execution of updates. 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 are 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 stay 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, resulting in her exclusion. Now the malicious party must submit a Freeze transaction to be allowed back in, which she can abuse 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.

E. 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 parameters 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). But instead of using lexicographic order after hashing the parties' identifiers to assign them the broadcast slots, 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 her 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 in 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 IV-C, we have

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

where x_i is the trapdoor of party $\mathfrak{p}_i$, which 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
$$\mathsf{dk}_{i,j} = \mathsf{ExtractDecKey}(\mathsf{pp}_j, x_i)$$
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 $\mathfrak{p}_i$ to permanently store x_i (which matches our claim of constant-sized decryption keys). In particular, $\mathfrak{p}_i$ need not permanently store $\mathsf{dk}_{i,j}$, since it can be computed on the fly as outlined above.

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 parameters are 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.

V. Benchmarks and Application

A. Benchmarking Results for FDBE

In this subsection, we present benchmarking results for our FDBE scheme from Section IV-C. We also present a detailed comparison of FDBE with other pairing-based BE schemes with distributed key generation listed earlier in Table I. We note that [46] and [45] essentially use the same underlying DBE scheme proposed in [51], so we just benchmark [51]. We use the BLS12-381 [30], [40] elliptic curve for the group and pairing operations, and AES-GCM for symmetric encryption/decryption. All measurements were made on a VM running RHEL 9 with an 8-core CPU (each core running at 2.4GHz) and 32GB memory.

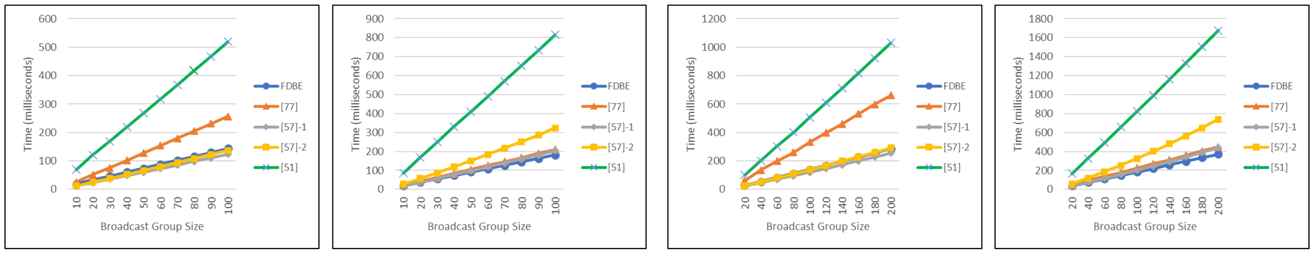
Scheme	Setup	n	|pp| (in KB)	|sk| (in KB)	|ct| (in KB)
[77]	Decentralized	10	17.27	2.30	0.85
		20	56.41	4.21	
		50	351.96	9.95	
		100	1157.03	19.53	
		200	4501.56	38.68	
		500	27660.15	96.05	
[57]-1	Trusted	10	4.73	0.19	0.74
		20	9.65		
		50	24.42		
		100	49.0		
		200	98.25		
		500	245.91		
[57]-2	Trusted	10	22.67	0.19	0.74
		20	83.09		
		50	494.96		
		100	1945.05		
		200	7710.67		
		500	48003.55		
[51]	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 II: 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. [46], [45], [51] essentially use the same underlying DBE scheme proposed concretely in [51], so we just benchmark [51].

n	[77]	[57]-1	[57]-2	[51]	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 III: 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 [77] for $n > 200$ as the process ran out of memory.

Component Sizes. In Table II, we concretely compare FDBE with the other schemes from Table I in terms of the sizes of their public parameters, decryption keys, and ciphertexts. The reported figures follow the asymptotic analysis in Table I. The size of public parameters for FDBE is significantly smaller than that of [77] and [57]-2 (even though the latter requires a trusted setup), and marginally larger than that of [57]-1 and [51]. 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 [57]-1 and [51] (illustrated subsequently). Finally, the decryption key and ciphertext sizes for FDBE are constant, and are either comparable to or smaller than those of the other schemes. Notably, [77] has non-constant key size (grows linearly with n) and $\approx 15\%$ larger ciphertext size than FDBE.



(a) Encryption Times ($n = 100$) (b) Decryption Times ($n = 100$) (c) Encryption Times ($n = 200$) (d) Decryption Times ($n = 200$)

Fig. 5: 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 [77] 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.

Key Generation. Table III compares the schemes from Table I 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, [57]-1,2 and [51] require a linear (in n) number of exponentiations for key generation. The key generation time for [77] 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. 5 compares FDBE with the other schemes from Table I 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 [77] ran out of memory for $n > 200$) and vary the target broadcast group sizes. We choose this setting because the encryption and decryption 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, [51] performs significantly worse than all other schemes due to it highly involved encryption and decryption algorithms. Overall, FDBE offers a desirable balance in terms of encryption and decryption performance. Notably, in comparison with [77] (which is the only other scheme with fully decentralized setup), FDBE supports *significantly faster* encryption *and* concretely faster decryption (this is because, unlike [77], FDBE avoids the heavy usage of target group operations for encryption and decryption).

B. 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 IV-C) in enabling confidential interoperability between private blockchain/DLT networks. We present an augmentation of Hyperledger Cacti [25], an open-source DLT interoperability framework, where networks built on Hyperledger Fabric [27] 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.

1) Hyperledger Cacti: Hyperledger Cacti is an open source project within Linux Foundation Decentralized Trust [15] 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 [7], [8]. Cacti supports protocols and provides reference implementations for networks to share (communicate) ledger data, atomically swap (exchange) and transfer assets on demand [26] (using community-agreed standards [53]). 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; this was created in Weaver, originally a separate interoperability framework, but now part of Cacti [14] (see Fig. 6(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 the 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 via consensus for the view data (Step 4) which is then independently validated by the destination network's peers (also via consensus). An authenticity proof typically consists of digital signatures (ECDSA for HLF). 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. 6(a), i.e., encryption in the source network, decryption at the destination client, and

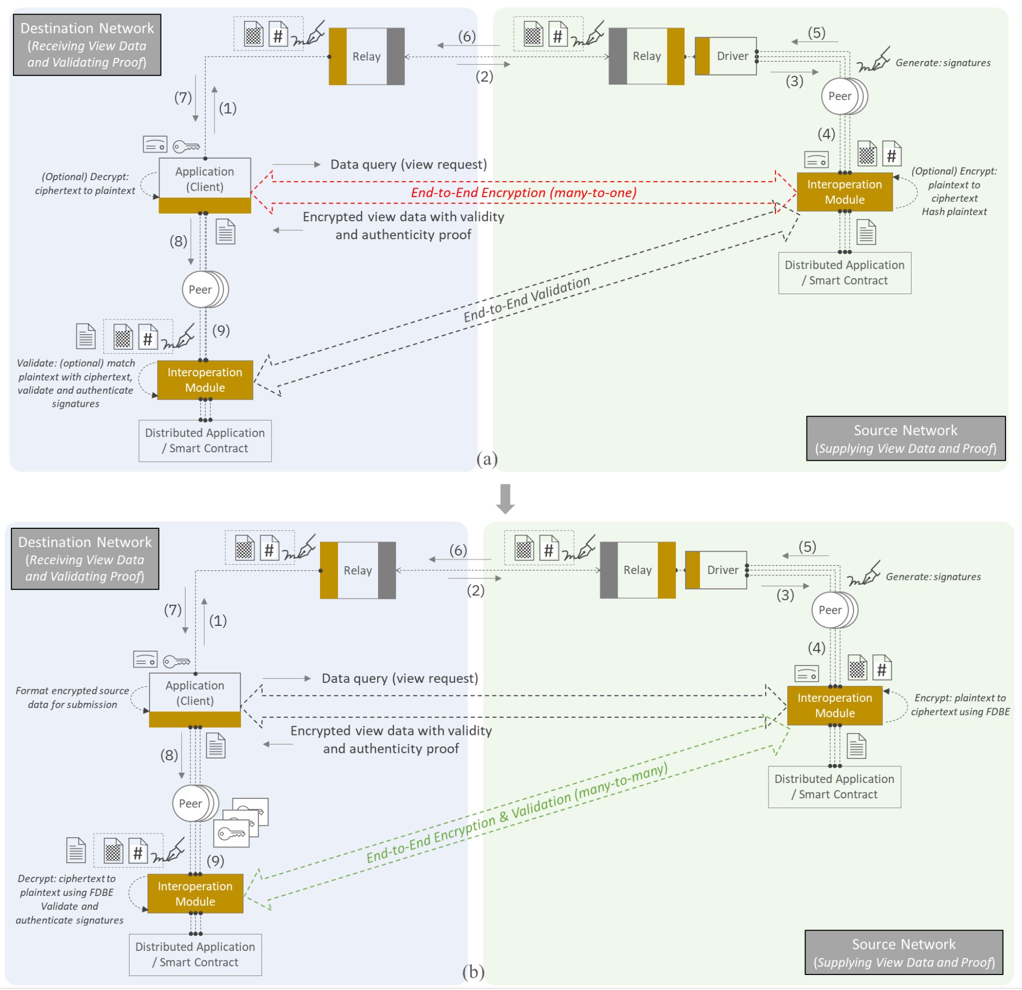

Destination Network
(Receiving View Data and Validating Proof)
Relay
(6)
(2)
Relay
Driver
(5)
(3)
Generate: signatures
Peer
(4)
(7)
(1)
Application (Client)
(Optional) Decrypt: ciphertext to plaintext
Data query (view request)
End-to-End Encryption (many-to-one)
Encrypted view data with validity and authenticity proof
Interoperation Module
(Optional) Encrypt: plaintext to ciphertext Hash plaintext
Distributed Application / Smart Contract
(8)
Peer
End-to-End Validation
(9)
Validate: (optional) match plaintext with ciphertext, validate and authenticate signatures
Interoperation Module
Distributed Application / Smart Contract
Source Network
(Supplying View Data and Proof)
(a)
Destination Network
(Receiving View Data and Validating Proof)
Relay
Relay
Driver
Generate: signatures
Peer
Format encrypted source data for submission
Application (Client)
Data query (view request)
Interoperation Module
Encrypt: plaintext to ciphertext using FDBE
Encrypted view data with validity and authenticity proof
Distributed Application / Smart Contract
Peer
End-to-End Encryption & Validation (many-to-many)
Decrypt: ciphertext to plaintext using FDBE Validate and authenticate signatures
Interoperation Module
Distributed Application / Smart Contract
Source Network
(Supplying View Data and Proof)
(b)

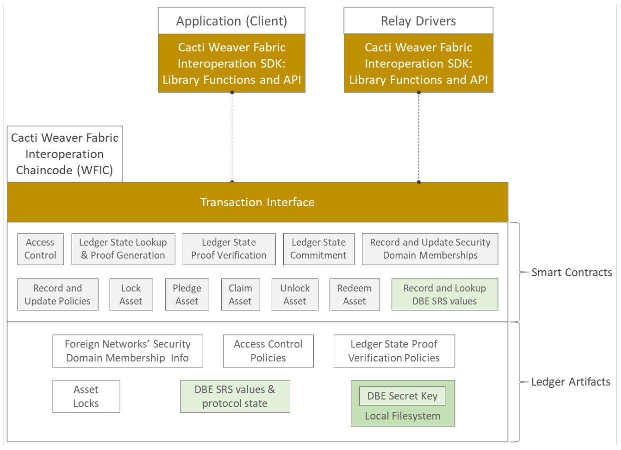
Fig. 6: Cacti Data Sharing Protocol (images adapted from Hyperledger Cacti Weaver RFCs [9]): (a) CN-PLAINTEXT when *optional* steps are skipped, and CN-PROXY when *optional* steps are performed; (b) Our implementation of CN-FDBE.

matching the plaintext with the ciphertext during the validation step in the destination network, are skipped.

Confidential Data Sharing in Cacti. Cacti also offers a mode that adds confidentiality against the communication channel, and hence, ensures 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); 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 decryption and before validation by the peers of the destination network [9]. However, as discussed in Section I, this protocol inherently relies on a centralized trust assumption; indeed, *there is no confidentiality guarantee against a dishonest (potentially malicious) client/proxy*.

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. 6(b)). While our implementation focused on HLF networks, the methods and APIs are portable to other DLTs (for e.g., Corda, Besu).

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* [11]. 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 [12]), which must be deployed on all the peers of a HLF's ledger (also known as *channel*) to make it interoperable with other Cacti-enabled networks [10]. Fig. 6(a) illustrates the *interoperation*



Application (Client)
Cacti Weaver Fabric Interoperation SDK: Library Functions and API
Relay Drivers
Cacti Weaver Fabric Interoperation SDK: Library Functions and API
Cacti Weaver Fabric Interoperation Chaincode (WFIC)
Transaction Interface
Access Control
Ledger State Lookup & Proof Generation
Ledger State Proof Verification
Ledger State Commitment
Record and Update Security Domain Memberships
Record and Update Policies
Lock Asset
Pledge Asset
Claim Asset
Unlock Asset
Redeem Asset
Record and Lookup DBE SRS values
Smart Contracts
Foreign Networks' Security Domain Membership Info
Access Control Policies
Ledger State Proof Verification Policies
Asset Locks
DBE SRS values & protocol state
DBE Secret Key
Local Filesystem
Ledger Artifacts

Fig. 7: Cacti WFIC augmented with FDBE Capability

module deployed 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 public parameters, sequentially update them, and then 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 [6]), 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 HLF's 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. 7. 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 IV-C). 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 view requests and validations – with wrapper functions that generate and decrypt FDBE-encrypted payloads respectively. See the full version of the paper [43] 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.

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. Details appear in the full version of the paper [43].

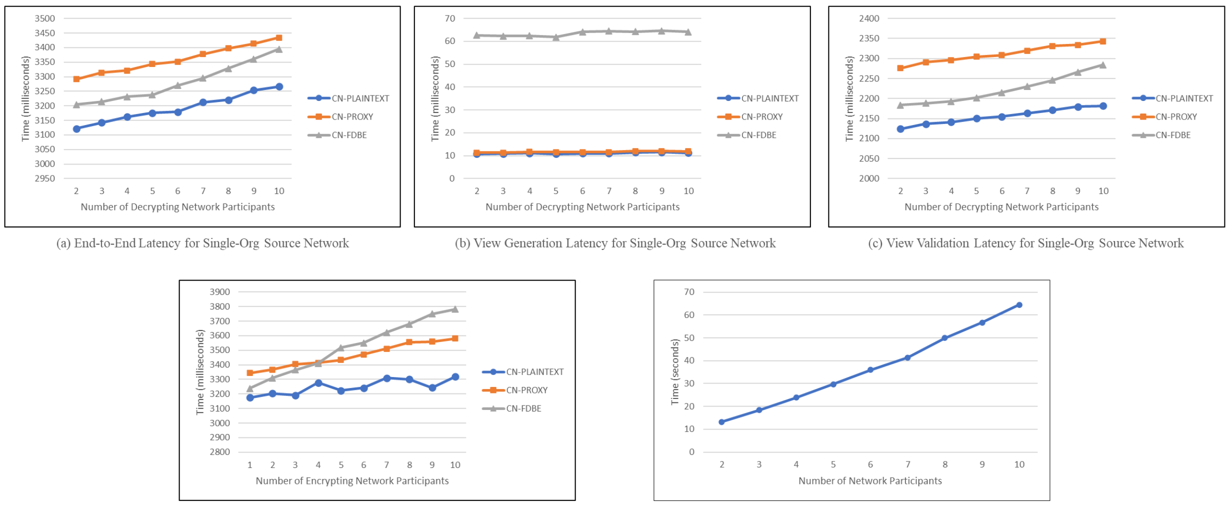
Bootstrap or Setup Phase. One or more organization members in the destination network trigger the initialization and validation of public parameters. If the network has N organizations (parties), the public parameters update and corresponding validation are triggered N times in sequence. Each update is triggered by a member of a different organization, and is idempotent (i.e., one organization may trigger an update exactly once), and update success depends on the previous steps having occurred in the right sequence. 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. 6(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.

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. 6). 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 contributions, which assume the necessity and practicality of cross-network data sharing (see Section I) and tackle potential attacks on the communication protocol. Furthermore, as shown in Section V-A, our FDBE scheme requires less storage compared to [77], 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 [77], as throughput varies with ciphertext size, which is $O(1)$ for both our 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

Fig. 8: CN-FDBE Performance: Measurements of Latency Across Two Networks (a-d) and Single-Network Setup (e)

and [77] (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 conducted 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. 8(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 compared to CN-PROXY and CN-PLAINTEXT (both with almost identical performance); see Fig. 8(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. 8(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 $\sim$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. 8(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, which incurs a higher latency for CN-FDBE compared to that of CN-PROXY. This is likely due to a less optimized implementation of FDBE compared to that of ECEIS. We leave further optimizing the implementation of FDBE as an interesting open question.

Fig. 8(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, not only does CN-FDBE provide a qualitative benefit over the state-of-the-art by enabling a fully-decentralized mechanism for confidential cross-network interoperability, but it also achieves performance gains for a large subset of practical configurations.

VI. 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 [15], (ii) the data sharing mechanisms in Cacti that our experimentation and benchmarking focus on are publicly documented [9], (iii) both Cacti and Fabric have been extensively used for experimentation and benchmarking in several prior works [27], [28], [53], [63], (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

[1] A Next-generation Smart Contract and Decentralized Application Platform. URL.
[2] DOGETHEREUM: We can do this Dogether! URL.
[3] Dtcc press releases: Project ion platform. URL. (Last accessed: Aug 22, 2022).
[4] Dtcc: Swift explores blockchain interoperability to remove friction from tokenized asset settlement. URL. (Last accessed: Jun 8, 2023).
[5] Global shipping business network. URL. (Last accessed: Aug 6, 2025).
[6] Hyperledger Cacti - Weaver Fabric SDK. URL.
[7] Hyperledger Cacti: Architecture. URL.
[8] Hyperledger Cacti: Project Scope. URL.
[9] Hyperledger Cacti: Weaver: End-to-End Confidentiality. URL.
[10] Hyperledger Cacti: Weaver: Fabric Interoperability Contracts. URL.
[11] Hyperledger Cacti: Weaver: Interoperation Modules. URL.
[12] Hyperledger Fabric: Smart Contracts and Chaincode. URL.
[13] Ibm food trust. URL. (Last accessed: Nov 30, 2021).
[14] Introducing Hyperledger Cacti, a multi-faceted pluggable interoperability framework. URL.
[15] Linux Foundation Decentralized Trust. URL.
[16] Marco Polo Network. URL.
[17] Project ubin: Central bank digital money using distributed ledger technology. URL. (Last accessed: May 29, 2025).
[18] TradeLens. URL.
[19] we.trade. URL. (Last accessed: May 13, 2022).
[20] Hashed time-locked contract transactions. *Bitcoin Wik*, 2020. (Last accessed: May 13, 2022). URL: https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts.
[21] Banque de france demos ledger interoperability in cbdc trials. *Finextra Editorial*, 2021.
[22] Zkporter: A Breakthrough in L2 Scaling. URL, 2021.
[23] Hqlax, j.p. morgan, ownera and wematch demonstrate a cross-ledger repo. *HQLAx Announcements*, 2022. (Last accessed: April 22, 2025).
[24] The interoperability of cbdcs across networks and currencies. *HSBC Report*, 2022. (Last accessed: April 22, 2025).
[25] Linux foundation decentralized trust - hyperledger cacti, 2025. Accessed on 09-Jan-2025. URL: https://github.com/hyperledger-cacti/cacti.
[26] 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.
[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

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

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

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

[38] Dan Boneh and Brent Waters. A Fully Collusion Resistant Broadcast, Trace, and Revoke System. In *ACM CCS 2006*, page 211–220, 2006. doi:10.1145/1180405.1180432.

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

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

[41] Ran Canetti. Universally Composable Security: a New Paradigm for Cryptographic Protocols. In *IEEE FOCS 2001*, pages 136–145, 2001. doi:10.1109/SFCS.2001.959888.

[42] 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.

[43] Angelo De Caro, Kaoutar Elkhiyaoui, Sandeep Nishad, Sikhar Patranabis, and Venkatraman Ramakrishna. Distributed broadcast encryption for confidential interoperability across private blockchains (full version). Cryptology ePrint Archive, Paper 2025/2237, 2025. URL: https://eprint.iacr.org/2025/2237.

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

[45] 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.

[46] 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.

[47] 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.

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

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

[50] 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.

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

[52] 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.

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

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

[55] 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.

[56] 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.

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

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

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

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

[61] 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. doi:10.1016/j.bushor.2022.06.002.

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

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

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

[65] Meta Platforms. WhatsApp | Secure and Reliable Free Private Messaging and Calling. URL: https://www.whatsapp.com/.

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

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

[68] Victor Shoup. Practical Threshold Signatures. In *EUROCRYPT 2000*, 2000. doi:10.1007/3-540-45539-6_15.

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

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

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

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

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

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

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

[76] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

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

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

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

[80] 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. URL: https://eprint.iacr.org/2019/1128.