The Coco Framework

Technical Overview

Published August 10, 2017

Contents

Executive Summary

Over the last few years, enterprises have come to realize how blockchain, the technology that powers public networks such as Bitcoin and Ethereum, can be used to streamline their own business processes. However, they have also found that most existing blockchain protocols fail to meet several key enterprise requirements—including acceptable transaction throughput and latency, confidentiality, effective governance, and computational efficiency (i.e., the energy cost for mining/proof of work). Efforts to adapt existing public blockchain protocols or to create new protocols to meet these needs have generally traded one required enterprise attribute for another—such as improved confidentiality at the cost of greater complexity or lower performance.

The Coco Framework is an open-source system that enables high-scale, confidential blockchain networks that meet all key enterprise requirements—providing a means to accelerate production enterprise adoption of blockchain technology. Coco achieves this by re-evaluating existing assumptions for public blockchain protocols in the context of a confidential consortium, where nodes and actors (including voting members and other non-voting participants) are explicitly declared and controlled. Based on this new set of requirements, Coco brings together the power of existing blockchain protocols, trusted execution environments, distributed systems, and cryptography to enable enterprise-ready blockchain networks that deliver:

- Throughput and latency approaching database speeds.
- Richer, more flexible, business-specific confidentiality models.
- Network policy management through distributed governance.
- Support for non-deterministic transactions.
- Reduced energy consumption.

It is important to note that Coco is not a standalone blockchain protocol; rather, it provides a trusted foundation with which existing blockchain protocols such as Ethereum, Quorum, Corda, or Hyperledger Sawtooth[1] can be integrated to deliver complete, enterprise-ready ledger solutions. Coco is designed to be open and compatible with any blockchain protocol.

This paper reexamines the requirements for a blockchain network in the context of a confidential consortium. It then introduces the Coco Framework, including network topology and system architecture, network creation and governance, transaction flow, and architectural variations. It also examines how scalability, confidentiality, and distributed governance are achieved, along with security considerations and how security risk can be mitigated. Finally, it discusses the current working implementation of Coco, including initial performance findings.

[1] Ethereum; Quorum by J. P. Morgan; Corda by R3; Hyperledger Sawtooth by Intel.

Facilitating Enterprise Blockchain Adoption

Blockchain technology is poised to become the next transformational computing paradigm. It promises to disrupt existing business processes, to reduce the friction of doing business, and to unlock new business models—especially shared processes across organizations. Given such benefits, in our rapidly evolving digital economy, it won't be long before blockchain technology is a key foundation for distributed enterprise and consumer applications.

Framing the Problem

Blockchain technology is already generating strong interest among enterprises across almost every industry. However, as enterprises look to apply blockchain technology to meet their business needs, they have come to realize that most existing blockchain protocols fail to meet key enterprise requirements in areas such as performance, confidentiality, governance, and required processing power. This is because they were designed to function—and to achieve consensus—in a potentially hostile environment, where anonymous, untrusted actors can execute transactions or propose block commits. As such, to prevent malicious behavior, transactions are "in the clear" for all to see, every node in the network executes every transaction, and Byzantine fault tolerant consensus algorithms must be employed. All of these "safeguards," while necessary to ensure the integrity of public blockchain networks, require tradeoffs in terms of key enterprise requirements such as scalability and confidentiality.

For instance, the public Ethereum network has an average processing rate of 20 transactions per second, with a typical transaction latency of around 10-20 seconds.[2] By contrast, the Visa credit card processing system averages 2000 transactions per second.[3] In addition, all Ethereum transactions, smart contract code, and state are in the clear—visible to anyone who joins the network.

Processing power required for public blockchain networks—and associated energy costs—are also prohibitive to enterprise scenarios. According to Digiconomist[4], the annual energy consumption of the Bitcoin network is equal to 14.96 terawatt-hours (TWh), with estimated annualized mining (transaction processing) costs of more than US$747 million. Put another way, the Bitcoin network consumes enough energy to power more than 1.3 million U.S. households.

There have been efforts to adapt public blockchain network protocols or to create new protocols to meet the needs of the enterprise. However, so far, these approaches have not met all enterprise requirements; most have required undesirable tradeoffs in terms of complexity or performance. For example, while approaches exist that leverage zero-knowledge proofs[5] or other advanced cryptography to hide transaction details, they are complex and require computational space and time tradeoffs making them unsuitable for high-volume applications, or they rely on a trusted source to bootstrap crypto material. Other schemes introduce rigidity because confidential channels require "commitments" of tokenized resources, making them inaccessible outside the channel while the channel is active.

---

[2] https://etherscan.io/chart/tx
[3] https://en.bitcoin.it/wiki/Scalability
[4] http://digiconomist.net/bitcoin-energy-consumption
[5] ZCash and Hawk are two such systems.

A Consortium-First Approach

Adapting existing, public blockchain technology to the needs of the enterprise (or, more accurately, to a consortium of enterprises) begins with re-examining the environment in which the blockchain must operate. In a public blockchain network, anyone can transact on the network, actors on the network are pseudo-anonymous and untrusted, and anyone can add nodes to the network—with full access to the ledger and with the ability to participate in consensus.

In contrast, in a consortium blockchain network, member identities and nodes are known and controlled. Actors are often equally mature, with robust and highly controlled IT environments, security policies, and other enterprise characteristics.

So how can we take advantage of these differences to meet the needs of the enterprise? Fundamentally, the trust relationship between the actors in a consortium need not be changed. However, if we can create a trusted network of physical nodes without requiring the actors that control those nodes to trust one another, we can control what code is run and guarantee the correctness of its output—thereby simplifying consensus and reducing duplicative validation. This is the approach that Microsoft has taken in developing the Coco Framework, which is described in the remainder of this paper.

The Coco Framework

The Coco Framework is an open-source system that enables high-scale, confidential blockchain networks. It meets all key enterprise requirements, providing a means to accelerate production adoption of blockchain technology. Coco achieves this through the use of ***trusted execution environments (TEEs)*** such as Intel's SGX and Windows Virtual Secure Mode (VSM), enabling the creation of a trusted network of physical nodes on which to run a distributed ledger.

With Coco, enterprises can benefit from:

- **Throughput and latency approaching database speeds**. Through its use of TEEs, Coco creates a network of trusted nodes that reduces the problem from Byzantine fault tolerance to crash fault tolerance. This simplifies consensus and thus improves transaction speed and latency—all without compromising security or assuming trust.
- **Richer, more flexible confidentiality models.** Coco uses industry standard authorization and authentication to safeguard data access. Transactions, smart contract code, and smart contract state can be processed in the clear yet revealed only to authorized parties, without requiring complicated confidentiality schemes.
- **Network policy management through distributed governance**. Coco provides a network constitution to express and manage consortium policies. Expression of network policies can be codified, and policies can be managed through standard ledger-like transactions.
- **Non-deterministic transactions.** With Coco, because the results of code execution can be trusted, smart contract code need only be executed by a single node in the network. When the network is configured in this manner, transactions can have non-deterministic results (e.g. by incorporating randomness in calculations) and can even interact directly with trusted external systems.
- **Reduced energy usage.** Coco improves energy consumption by eliminating computationally intensive consensus algorithms, such as proof of work.

Conceptual Overview

It is important to note that Coco is not a standalone blockchain protocol; rather, as illustrated in Figure 1, it provides a trusted foundation that delivers efficient consensus algorithms and flexible confidentiality schemes—a framework with which existing blockchain protocols such as Ethereum, Quorum, Corda, or Hyperledger Sawtooth can be integrated to deliver complete, enterprise-ready ledger solutions.

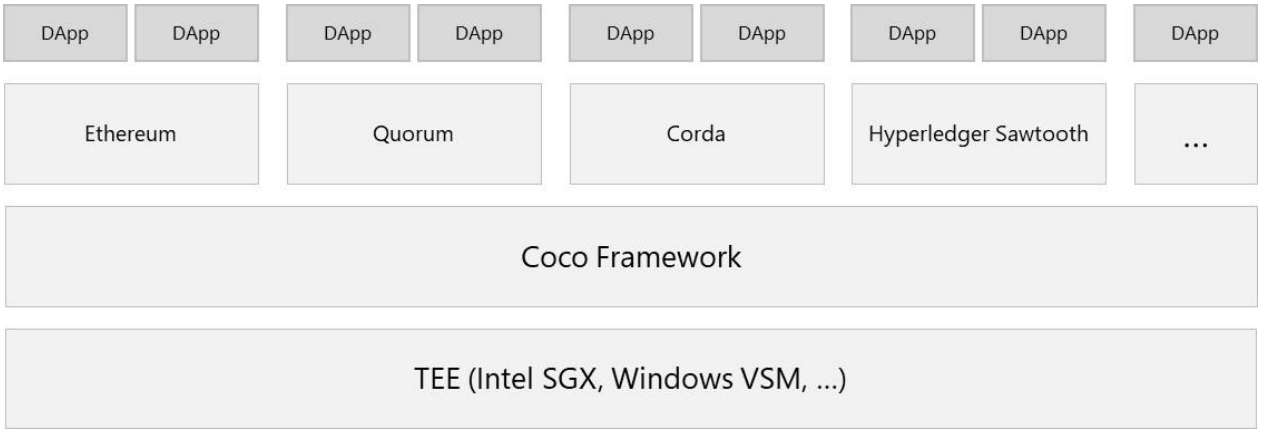

Figure 1: High-level overview of a Coco system.

Coco was designed to provide secure, reliable base components that any blockchain protocol can use. Figure 2 illustrates the separation/overlap between Coco and a blockchain protocol, with the Coco components shaded in green. At its core, Coco:

- Implements a consistent, distributed, persistent store (such as a key-value store) that is replicated across TEEs—containing both the application (business transaction) ledger and the Coco administrative ledger used for network policy management. While there are logically two ledgers, both are recorded within a single store to maintain relative ordering across all transactions in the network.
- Provides secure node-to-node and application-to-node communication.
- Enables arbitrary confidentiality models with easy-to-use primitives.
- Provides a codified governance model to support arbitrary, distributed policy management.

Coco relies on the specific blockchain protocol for the distributed ledger model, as well as core transaction and smart contract code processing.

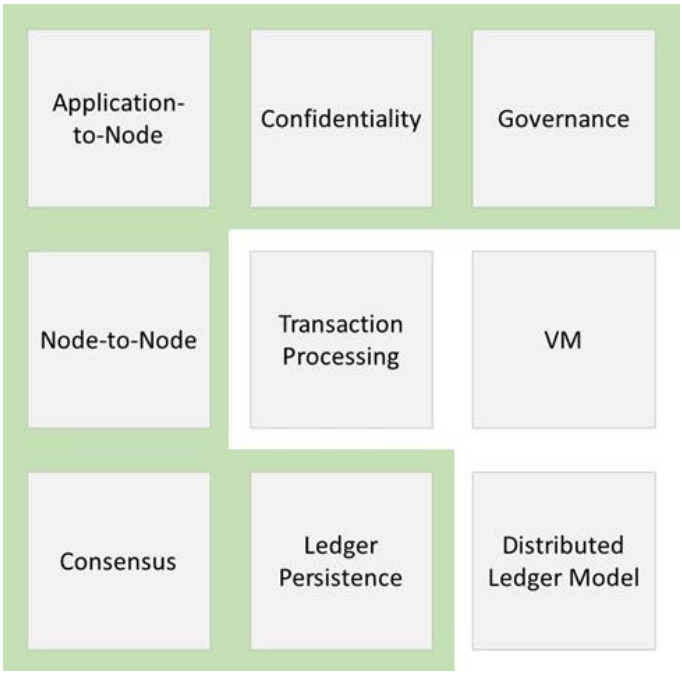

Figure 2: Logical components of a blockchain protocol (Coco components are shaded in Green).

High-Level Architecture

A Coco system is a distributed network of trusted ***validating nodes (VNs),*** each of which run the Coco Framework and the integrated blockchain protocol. VNs accept transactions and participate in the network's consensus algorithm. Depending on the chosen consensus algorithm, one or some VNs process transactions and execute smart contract code.

Loosely, the VNs in a Coco network are similar in function to the mining nodes in a public blockchain network—though the acts of "mining" and communication in a Coco network are very different. Unlike public blockchain networks, each VN in a Coco network is fully trusted because VNs can verify the identity of all other VNs and validate the code they run through TEE attestation at runtime.

As illustrated in Figure 3, a VN can be decomposed into several logical sub-components. These are divided across the host (the potentially untrusted hypervisor and operating system of the node) and the enclave, which is a protected, secure container within the TEE. All sensitive components of the system run inside the enclave and are responsible for maintaining integrity, confidentiality, and security of the ledger.

- **Coco interface (host)** – the host process that implements the transport layer interface to which the client and other VNs connect. This host process also interfaces with the operating system and other external services on behalf of the enclave. (While the host itself is required to maintain availability of the VN and persistence of the data stored in the ledger, it has no responsibility for maintaining integrity, confidentiality, or security of the ledger.)
- **Coco core (enclave)** – the interface between the host and all other functionality within the enclave.
- **Persistent store (enclave)** – the single, globally ordered, on-disk representation of the application ledger and the Coco administrative ledger. (There is an in-memory copy for fast access; durability is achieved via a persistent store.)

- **Coco configuration state (enclave)** – the Coco state machine implementation that supports creation of the network constitution and all subsequent updates.
- **Blockchain core and adapter (enclave)** – the logic to process transactions and execute smart contract code, as well as the glue to integrate the blockchain protocol into the rest of the Coco system.

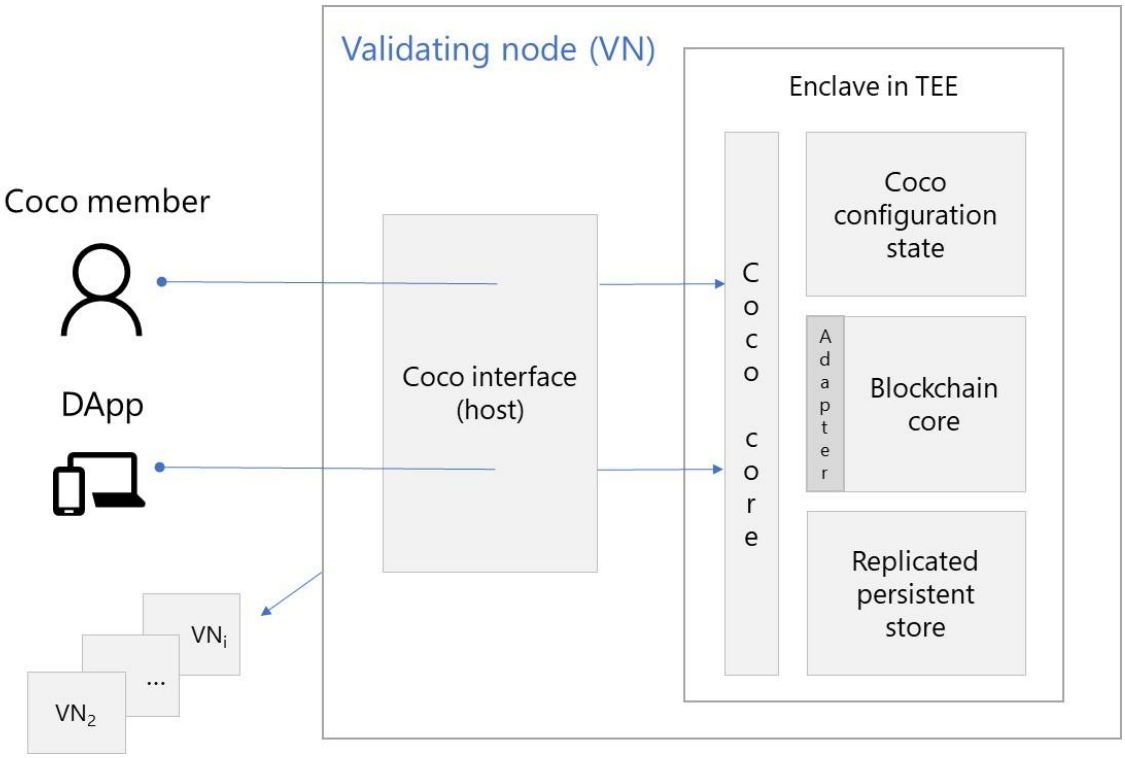

Figure 3: Coco architecture.

Actors and Identity

There are two types of actors in a Coco network, members and participants. At a high level:

- Members are the governing bodies of a consortium, with collective control over who can transact on the network and its governance—including network membership, the code that runs in the TEEs, and the definition of network policies. Members run VNs and can transact on the network. Members also define the initial configuration and setup of a Coco network and, throughout the lifetime of the network, can propose changes to its configuration and vote on changes proposed by other members. Further, each member receives a share of the network's shared network key (described later), which allows a quorum of members to collaboratively restore the network in the event of a catastrophic failure. In a consortium of banks, members could be large, global, systemically important financial institutions (GSIFIs).
- Participants, unlike members, cannot vote and thus have no operational control over who can directly access the network or its governance. Participants are determined by the network's members and, like members, participants can transact on the network. They also can, and likely do, run their own VNs. In a consortium of banks, participants could be a group of domestic systemically important banks (D-SIBs) that transact between themselves and the GSIFIs.

It's worth noting that, depending on the purpose and policies of the consortium, a Coco network doesn't necessarily need to have both types of actors. For example, a consortium may decide that all parties within it should be members, with equal responsibilities and permissions. Similarly, each actor could have both member and participant identities, as a means of separating business and administrative transactions.

Every member and participant has a private/public key pair ($PrivKey_{Mi}$, $PubKey_{Mi}$ and $PrivKey_{Pi}$, $PubKey_{Pi}$, respectively) and is identified in the Coco network with an organization-wide X.509 certificate. An intermediate certificate can be used to uniquely identify the organization and all users within it, abstracting actor-specific certificate policies and roll-over from the Coco system. Within the organization, to maintain fine-grained control, different departments that are participating in different consortiums (and thus different Coco networks) can have different certificates to minimize cross-network access. This allows for easy integration with existing corporate systems and public-key infrastructures (PKIs).

End users, such as a bank's customers, do not have an identity in the Coco network and cannot transact on the network directly. Instead, they interact with it through the front-end systems of members or participants.

In a typical scenario, as illustrated in Figure 4, end-users are authenticated and authorized using existing enterprise identity management mechanisms, such as Active Directory (AD) or Azure Active Directory (AAD). The end user submits a transaction to a front-end system. The front-end system then submits the transaction to the Coco network using the member's or participant's Coco identity ($PrivKey_{Mi}$ or $PrivKey_{Pi}$), which is stored in a local or cloud-based key management system, such as a hardware security module (HSM).

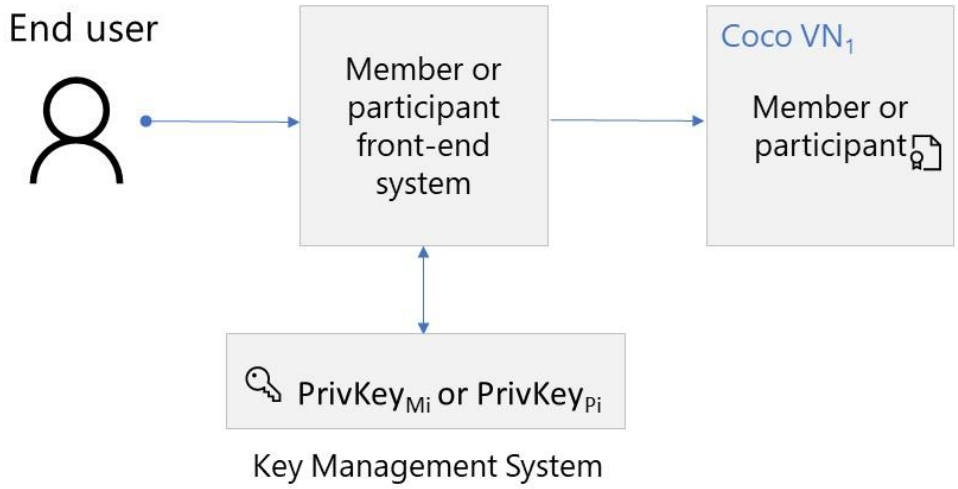

Figure 4: End user transaction flow.

Network Creation and Governance

Coco describes and manages network policies through a network constitution (described below) and a process of distributed voting. This allows consortium rules and agreements to be codified in the system, similar to how contracts are codified in smart contract code.

Any member can start a network by bootstrapping the first VN. After that, proposals for changes to the Coco network constitution (such as adding or deleting a member) can be submitted by any member for a vote. Like application transactions, such Coco administrative tasks are submitted as transactions to the Coco network.

Network Constitution

In a Coco network, the network constitution is the complete expression of network policies. At a minimum, these include the membership list, VN list, code manifest, TEE manifest, and voting policies.

- The ***membership list*** is a list of all approved actors (members and participants) in the network. A member, M_1, is represented by its private/public key pair ($PrivKey_{M1}$, $PubKey_{M1}$). If M_1 to M_i are members in the consortium, the member list would be represented by the public key set {$PubKey_{M1}$, … $PubKey_{Mi}$}. Only each member's public key certificate is uploaded into the Coco network. Each member's private key remains in the originating key management system, which can be on-premises or in the cloud. Participants are handled in the same way as members.
- The ***Code manifest*** is the specification of all approved code that can run within the Coco network—including but not limited to Coco Framework versions, blockchain protocols, and specific blockchain protocol versions. Any one code identity is represented as a single measurement.
- The ***TEE manifest*** is the specification of all approved trusted execution environments—hardware or software. VNs in a Coco network can run different TEEs, provided that all TEEs employed are acceptable per the TEE manifest. In such a mixed TEE environment, network security is only as strong as the weakest TEE mechanism employed.
- The ***VN list*** is a list of all approved validating nodes in the network. A VN is represented in the Coco network by a signed tuple that includes the member's identity, the address (either DNS name or IP address) of the physical node, and the public key corresponding to the private key generated within the enclave to uniquely identify the VN/secure enclave pair. For the first VN of the first member, VN_{M1_1} = {$PubKey_{M1}$, address of VN_{M1_1}, $PubKey_{VN\_M1\_1}$}$S_{PrivKey\_M1}$.
- Any change to the network constitution, such as additions or removals to the membership or VN lists, must be done through a voting process (described later). The criteria for a vote to be accepted is known as the ***voting policy***, and is also expressed in the network constitution. Options for a voting policy can include unanimous, M of N, one of N, etc. Similarly, policies on whether vote submission can be automated if it meets certain criteria can be expressed. For example, vote submission could be automated so that every member could add a specified number of VNs, provided that the VNs are running majority-endorsed code.

These are the fundamental policies a Coco network implements; we plan to support additional policies over time.

Bringing up the Network

Any member can start a network by bootstrapping the first VN and uploading a genesis network constitution. The member who starts the network has no extra privileges compared to subsequent members who join the network.

Bootstrapping a VN

To bootstrap a VN, a member or participant performs the following steps:

1. Sets up a server or virtual machine with a TEE and installs a Coco Framework-supported operating system.
2. Installs the Coco Framework and desired blockchain protocol. The software is installed on the server, which in turn creates an enclave in the TEE to run sensitive Coco and blockchain functionality. The TEE generates an attestation of its hardware/software mechanism and the code running inside the enclave, which the actor uses to verify that all is as expected. A private/public key pair is generated within the enclave and the public key is included as part of the attestation to represent the enclave's identity.
3. Uploads the network constitution to the VN.
4. Establishes a secure communication channel between each front-end application and the VN.
5. Uploads its private/public key shares, which are used for data integrity and confidentiality (described later).

The above process for bootstrapping a VN is the same whether it is the first VN in the network or a subsequent VN. Subsequent VNs will discover other VNs in network using the VN list and attempt to establish secure node-to-node connections with them. Secure connections are only established with other VNs if they are in the VN list and the TEE for each other VN can provide a valid attestation.

Member, Participant, and Node Management

After the network is up and running, any existing member can propose a vote to add a new member or participant; add a new VN belonging to an existing member or participant; or remove a member, participant, or VN.

Adding a New Member or Participant

Multiple members and/or participants can be defined in the genesis network constitution, in which case they do not need to be added after the network is up and running. New members or participants who are not defined in the genesis network constitution can be added by existing members.

Regardless of whether a new member or participant is being added, the process is the same. Here's how the process works for a new member:

1. The new member, M_J, creates a VN using the process described above. To do this, the new member must receive a copy of the current network constitution to use during bootstrap from an existing member. This happens outside of the Coco system today, through whatever secure means the members communicate. (While having a copy of the network constitution does not mean a rogue

entity can join the network, the network constitution does contain sensitive information, meaning care should be taken in sharing it with others.)

2. After the new member's VN is ready to join the network, an existing member M_i proposes a vote, providing M_J's identity and M_J's first VN to admit to the network: {$PubKey_{Mj}$, VN_{Mj_1}}. If the vote is successful, the member, M_J, is added to the membership list and its node (VN_{Mj_1}) is added to the VN list.
3. After the member is admitted to the network, it will receive the current network constitution and blockchain state from another member VN and ensure the constitution matches its own version.

To simplify the above process, on bootstrap (Step 1 above), a new member's VN can receive the current network constitution from the network itself to avoid passing it outside of the system. In this case, in Step 3, M_j must have a chance to approve the network constitution after the VN is setup, but before the VN downloads data or processes transactions, to avoid the case where M_j does not approve of the constitution at the time of M_j's admission.

Technically, the addition of an actor (member or participant) and its first VN can be decoupled if the new actor wants to be added to the membership list before its first VN is ready; however, in this case, a second voting proposal must be made through another member's infrastructure to add the new actor's first VN to the VN list.

Similarly, if the new actor bootstraps its first VN too early, that VN will be rejected when initiating connections with existing VNs in the network because the new actor's VN is not yet in the VN list. The lookup and connection attempt will need to be re-initiated after the vote has completed and the new actor's VN has been added to the VN list.

Adding New VNs for Existing Actors

Similar to adding a new member, a member may propose a vote on the addition of a new VN belonging to an existing actor. The proposal can be made by any actor, not just the one who owns the VN. The voting process is the same, and the new VN cannot join the network until the vote is successful.

Removing Actors or VNs

The process to remove an actor or VN is also similar. The actor or VN is not officially removed from the membership list or VN list until voting is completed and the vote is processed, meaning that the actor or VN will still have access to the network during that time.

Voting

Any member can propose a change to any of the policies specified in the network constitution for a vote, in a manner similar to how members and VNs are added or deleted through voting. Members can also propose values for policies that were not specified in the genesis network constitution, provided that the policy is supported by the Coco system. Similarly, members can propose to remove policies from the network constitution. Updates to the Coco Framework code and blockchain protocol code are also proposed and voted on. Even the voting policy itself can be changed through this process. (A vote to change the voting policy would use the old voting policy; if the vote succeeds, the next vote will use the new voting policy.)

When a proposal for a change to the network constitution is submitted, all existing members are notified and must vote. Each member gets one vote, regardless of the number of VNs it has in the network. Each member's vote is a Coco administrative transaction and, as such, is submitted, processed, and finally recorded in the Coco administrative ledger.

After a quorum of members have voted, the Coco system determines the result of the vote based on the designated voting policy. Members can vote at any time, meaning that the delay before the result of a vote is determined depends entirely upon how quickly members vote and the voting policies expressed in the network constitution. Changes to any network policy do not take effect until the quorum has finished voting and the change approved.

Transaction Workflow

After a Coco network is established, it can accept transactions from members or participants. There are two types of transactions:

- Application transactions, the main set of business transactions through the network.
- Coco administrative transactions, such as adding a member.

The Coco Framework uses similar workflows to receive and process both types of transactions, including how consensus is achieved. (Recall that, while there are logically two ledgers, both are recorded within a single store to maintain relative ordering across all transactions in the network.)

Regardless of transaction type, Coco uses secure communication channels for all application-to-node and node-to-node traffic to protect confidentiality. Coco also supports encryption and authentication at the application layer instead of the transport layer.

Application Transaction Workflow

Figure 5 illustrates the workflow for an application transaction:

1. Before sending transactions, an application (such as an Ethereum DApp) establishes an authenticated, encrypted connection with a VN in the network. The application can establish the connection directly with the VN or through an intermediate front-end system, depending on the security and key management design goals for the DApp and the front-end.
2. When the Coco Framework receives a transaction, it decrypts it, then routes it to the appropriate blockchain adaptor for the protocol. (The connection is accepted by the host, but is decrypted and processed inside the enclave. Through this approach, interactions at the networking and operating system layers on the host side of the VN take place without visibility into the data payload or the ability to tamper with it.)
3. The adapter starts a transaction set and interacts with components of the blockchain protocol sub-system (such as block generation logic and Ethereum Virtual Machine, or EVM) to process the transaction.
4. The adapter commits the transaction set via a distributed store, which persists and replicates all state changes across the network using the configured consensus algorithm. Processing of a single transaction may result in several state changes.

The Coco Framework supports batching of application transactions, however, this trades-off higher throughput for higher latency. When the adapter processes a batched set of transactions, it collects all updates in the transaction set before committing them to the distributed store.

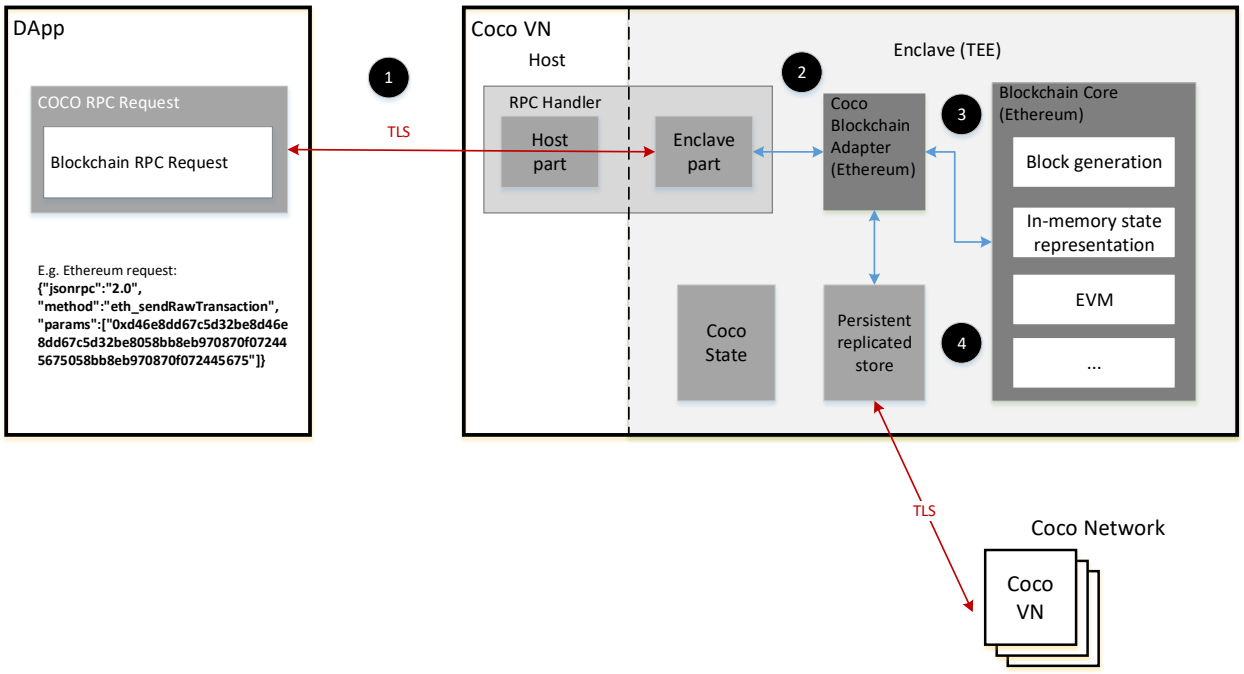

Figure 5. Workflow for an application transaction.

Coco Administrative Transaction Workflow

The workflow for processing a Coco administrative transaction is similar to the above; the only difference is that the transaction is handled by the portion of the Coco Framework that handles governance instead of the blockchain core.

Achieving Consensus

In a Coco network, consensus is required for all updates to the distributed store, including application transactions, smart contract state, and administrative transactions. While consensus is a fundamental aspect of any distributed network, compared to public blockchain networks, a Coco network is unique in that every VN fully trusts every other VN in the network.

Because of this, no need exists to defend against Byzantine faults, such as malicious messages, and blockchain updates that do not conflict with existing state maintained by a VN can be unequivocally accepted. The end result: Coco does not require wasteful, compute-intensive algorithms for proof-of-work, potentially unfair proof-of-stake algorithms, or latency-inducing time-bound algorithms. Defense-in-depth measures to mitigate the risk of TEE compromise or hardware errors in execution are covered later in this document, under Security and Compromise.

The Coco Framework is designed to support pluggable consensus algorithms—with plans to initially integrate Paxos-like consensus algorithms and Caesar consensus, an algorithm from Microsoft Research. Coco networks can be built with other consensus algorithms; to achieve efficient agreement and maximum throughput, ideal implementations should take advantage of the fact that a message received over a secure channel from a trusted VN is itself trusted.

Regardless of the consensus algorithm employed, consensus can be achieved as quickly as durability and serializability requirements are met for the algorithm. Distributing transactions throughout the network can be performed using multi-cast, broadcast trees, or any other protocol deemed efficient for the size and topology of the physical Coco network.

Paxos Consensus

With Paxos consensus, one VN is elected the leader. All other VNs are followers that simply accept transactions and forward them to the leader for processing. After the leader processes the transaction, thanks to the use of TEEs in Coco, followers can simply accept the transaction results from the leader.

Paxos consensus (and Paxos-like algorithms, such as Raft) are highly efficient for a smaller number of nodes, and are traditionally limited by the median latency between nodes. As such, the suitability of Paxos-like consensus algorithms for a Coco network is dependent on the number of VNs—which, in turn, depends on the number of actors in the consortium and the number of nodes per actor.

In future versions of Coco, node scalability could be addressed by adopting distributed systems research such as Flexible Paxos—the simple observation that it is not necessary to require all quorums in Paxos to intersect. It is sufficient to require that the quorum used by the leader election phase will overlap with the quorums used by previous replication phases. This allows Coco to trade slower leader elections (an uncommon event) for faster replication (a common event).

Node scalability could also be addressed through distributed transaction support, using approaches such as optimistic commits and dependent commits on followers. This allows all nodes, not just the leader, to process transactions, spreading the computational load across the Coco network.

More information on Paxos can be found here and more information on Raft can be found here.

Caesar Consensus

Caesar consensus is a new consensus algorithm for architecting secure and efficient consortium blockchain networks. It leverages the cryptographic properties of a blockchain to enable nodes in the blockchain network to reach consensus on a valid ledger. Caesar consensus supports flexible fault tolerance models for network nodes—including crash, Byzantine, a mix of both, or near-total compromise (i.e., all-but-one-node compromises).

In a Coco network, Caesar consensus can be used with traditional fault tolerance algorithms (such as Paxos) to help provide additional security and compromise detection, such as in the event of a TEE compromise. Under such a combined approach, rather than unequivocally accepting transactions from a leader node, follower nodes store small bits of cryptographic information (called heartbeat transactions) within the distributed ledger to enable distributed consensus, which also tracks progress and keeps the leader accountable. If misbehavior is detected, follower nodes can come together to eject the leader from the network, upon which another leader is elected. (Recall that, in a Coco network, the leader can only act maliciously if its TEE is compromised. Caesar consensus detects such a compromise and initiates a failover protocol to start a new leader, which in turn addresses the compromise.)

More information on Caesar consensus can be found in Section 3.4 of this paper.

Secure Communication

Coco enforces secure application-to-node and node-to-node communication channels, both of which use mutually authenticated TLS connections that terminate in the enclave. (Technically, there are two application-to-node communication channels—one for application transactions and another for Coco administrative transactions.)

This ensures that only valid members and participants can submit, process, and view network transactions, and that only valid VNs can join the network. Similarly, because data is encrypted across all connections and is only in the clear within the enclave, host-level access to a VN does not allow for inspection of network traffic.

Confidentiality and Integrity

A Coco network's persistent state—meaning the state that's written to durable storage—includes three types of data: application transactions, smart contract state, and administrative transactions. All are protected to ensure network confidentiality and data integrity—i.e., to ensure that only valid actors on the Coco network can access the data, and that the data is unaltered. Coco also supports fine-grained data access control, as required to accommodate business requirements for more granular confidentiality—such as 2+ party confidentiality.

Network Confidentiality and Data Integrity

At a minimum, Coco protects ledger entries to ensure that no entity outside of the network can view or manipulate the persistent state. Through the use of threshold encryption (described below), Coco can achieve network confidentiality and data integrity without granting any single actor full control.

Within an enclave on the VN, data can be processed in the clear to simplify computation and increase performance. However, given that the TEE has limited secure memory, data that is written to the persistent store must pass through the untrusted host before it is eventually written to disk. Coco addresses this by writing-out all data from the enclave in encrypted and integrity-protected form using authenticated encryption. This ensures that the data can only be accessed through the Coco Framework, thereby ensuring proper access control, protecting against unauthorized direct access to the host or disk, and ensuring that tampered data is detected.

The encrypted data on disk can be backed-up to other on-premises or cloud storage, as required to ensure high availability and protect against malicious (or unintended) data deletion.

Encrypting Persistent State

To encrypt persistent state, Coco generates a ***symmetric session key*** ($Key_{session}$) within the enclave of the VN that is designated as the leader within the consensus algorithm. The key is generated across a defined epoch (a fixed period of time set by members in the network constitution), with the specific duration of the epoch determined by the consortium after weighing the tradeoffs between increased security and the operational overhead of more frequent key generation. Other changes to network state, such as actor addition or removal, will also force a transition of epoch.

The symmetric session key is itself encrypted and written to the Coco administrative ledger for persistence. The symmetric session key also is maintained in memory for a fixed period of time (configurable in the network constitution); if not in memory, the key is retrieved and decrypted as needed.

The process that Coco uses to encrypt and persist session keys leverages an entirely independent asymmetric ***threshold cryptosystem*** established expressly for this purpose. Under this approach, N actors exchange material to establish an M of N ***threshold encryption scheme*** as follows:

1. Each actor generates a private key share ($PrivKeyShare_{Mi}$ or $PrivKeyShare_{Pi}$), creates a corresponding public key share ($PubKeyShare_{Mi}$ or $PubKeyShare_{Pi}$), and uploads both to its VN. If desired, to avoid having to move the private/public key share between systems, this step can be done within the Coco system on the VN.

2. The VNs each share their actor's public key share ($PubKeyShare_{Mi}$ or $PubKeyShare_{Pi}$) with other VNs. The private key share ($PrivKeyShare_{Mi}$ or $PrivKeyShare_{Pi}$) remains *only* on each actor's nodes.

3. Each VN then uses the shared public key share material ($PubKeyShare_{Mi}$ or $PubKeyShare_{Pi}$) to form a single epoch public key—called the ***public shared network key***—to encrypt the symmetric session key ($Key_{session}$). A simple derivation for an N of N scheme could be to multiply all public key shares together: $\prod_{i=1}^{N} \mathrm{PubKeyShare}\ Z_i$. More sophisticated derivations can be used for M of N schemes where M < N.

A threshold encryption of the symmetric session key allows for:

- **Access Control:** A malicious VN cannot decrypt the data on its own; it instead needs to acquire partial decryptions from other M-1 other actors' VNs in the network before such an attempt can be made.
- **Business Continuity and Disaster Recovery:** In the event of a catastrophic failure, actors can work together to decrypt data outside of Coco by combining their decrypted shares of the symmetric session key ($Key_{session}$). (With a sufficiently large M, it is unlikely that enough actors could maliciously collude to decrypt data under other, non-warranted circumstances.)
- **Auditing:** Auditors can request decrypted shares of the symmetric session key without needing the full private shared network key or relying on a single actor to provide the data.

Decrypting Persistent State

M of N actors are needed to decrypt the symmetric session key ($Key_{session}$), where M is configurable from 1 to N. To decrypt, the leader must request and acquire M of N decrypted shares of the symmetric session key. The VN can then combine the decrypted shares to form the complete decryption of the symmetric session key. It is important to note that the private key shares ($PrivKeyShare_{Mi}$ or $PrivKeyShare_{Pi}$) are never combined within the Coco network, thus never revealing the full private shared network key.

It's worth noting that, while M can equal N, or can even be 1, a typical Coco implementation will require that M be at least ½N+1 to ensure that confidentiality will remain protected provided a majority of actors do not collude. This also ensures that the network can continue to operate as long as it can maintain connections with VNs representing at least M+1 actors, enabling it to continue operating even when the VNs of one or more actors are withdrawn or partitioned from the network.

2+ Party Confidentiality

Limiting transaction visibility to valid actors on the network may not be sufficient when two or more actors within the network need to transact privately, such as for regulatory reasons. To provide the ability for an actor to designate a transaction between itself and one or more other actors as confidential, support for such an expression should be present (or added) in the integrated blockchain protocol.

Through the use of TEEs, this turns into a standard data access control problem. Once again, the transactions can be processed in the clear within the enclave, and then stored in an encrypted format outside of the enclave. The VNs then only allow the actors involved in the transaction to see it. The same model can apply to smart contract code and smart contract state, not simply transaction data. We discuss how this can be implemented for Ethereum later in this paper, under Functional Prototype and Demos.

Additional security measures may be needed for certain types of transactions. Nothing in Coco prevents an application from using additional confidentiality techniques, such as homomorphic encryption, to

encrypt transactions before they are submitted to the network; this simply becomes another layer on top of Coco-specific network and 2+ party confidentiality.

With the above 2+ party approach, the data resides on all VNs even though it is only visible to some. In some regulatory scenarios, however, transaction data simply cannot reside on all nodes in the network. To address this, in addition to standard 2+ party confidentiality as described above, point-to-point models can be integrated into a Coco system.

Security and Compromise

Coco VNs trust their peers, enabling highly efficient, scalable, and performant transaction processing. However, this approach presents a potential vulnerability. If the TEE on a VN is compromised, an attacker could potentially access private data and post spurious transactions. This is unlike traditional blockchain implementations, in which the compromise of a single node is a localized failure that does not affect the broader network. In considering this, there are a number of measures that can be taken to substantially mitigate the risks associated with TEE compromise:

Compile-Time Techniques

Compile-time techniques can be employed to eliminate classes of bugs that could be used to compromise the TEE, as described in Section 7 of VC3: Trustworthy Data Analytics in the Cloud using SGX.

Minimizing the Trusted Code Base

Minimizing the potential attack surface reduces the risk of compromise. This can be done by factoring the code running within the enclave into two parts:

- **Manager enclave.** The manager enclave is responsible for sensitive operations, including private key management, cryptographic processing, and governance logic. This code is bounded in size and need not vary with each blockchain integration. Furthermore, it can be formally verified and heavily audited.
- **Worker enclave.** The worker enclave hosts the blockchain protocol code and calls the manager enclave for sensitive operations. If the worker enclave is compromised, an attacker could access any plaintext data that resides in the worker enclave's memory, but could not directly access keys, independently decrypt data on disk, or propose or approve changes to the network.

Alternatively, software-fault isolation techniques can be employed within a single enclave. Not only is such an approach practical and low-overhead, but it prevents the leakage of keys in a single enclave. More information on such an approach is provided in A Design and Verification Methodology for Secure Isolated Regions, a paper from Microsoft Research.

VN Lease

In addition, the network can require that VNs obtain a lease that is periodically renewed. The renewal process can require that the VN's TEE be reset to a pristine state, eliminating any toehold that an attacker might have been able to gain. VNs that fail to renew their leases are quarantined from the network, preventing them from accessing data that is added to the blockchain after the quarantine takes effect.

Consensus Variations

Finally, it is possible to have VNs sample transactions received from other VNs and synchronously validate them before commitment. This would enable the rapid detection of a malicious VN without major performance tradeoffs. Similarly, a separate VN could asynchronously validate all transactions.

Functional Prototype and Demos

To demonstrate viability and inform its v1.0 design, Microsoft created a functional prototype of the Coco Framework and used it to create demos that showcase scalability and confidentiality.

Functional Prototype

The Coco Framework architecture is logically divided into several components: persistent distributed store with a consensus algorithm, secure communication channels between application-to-node and node-to-node, encryption schemes for confidentiality, and distributed governance. The functional prototype has the following implementation characteristics:

- A persistent, distributed key value store based on a RAFT consensus algorithm.
- Intel SGX hardware for its TEE.
- Secure application-to-node and node-to-node communication via mutual TLS authentication. (This was implemented using a modified version of mbed TLS, an open source TLS implementation.)
- A simplified network constitution that supports only member and node management.
- A version of an existing Ethereum protocol implementation for transactions and smart contracts, with minor modifications for integration with Coco.

Microsoft expects to make many of the components within the Coco Framework pluggable to support industry-specific requirements—including but not limited to consensus algorithms and communication protocols.

Demos

Using the prototype, Microsoft deployed a multi-node Coco network to assess scalability and showcase confidentiality.

Scalability Test

To assess scalability, Microsoft selected 2000 transactions from the public Ethereum network to feed through the prototype. In general, transaction rates were roughly 100 times faster than for a non-Coco-enabled protocol.

Specific results of the scalability test include the following:

- Throughput of around 1600 transactions per second. *(Throughput is defined as the total number of transactions processed and committed per unit of time. The speedometer in the demo displays total number of transactions per second.)*
- Latencies in the low hundreds of milliseconds, as measured with a 100-transaction batch size—representing the average blockchain block size. *(Latency is defined as the total time from when the transaction is received to when it has been committed by all nodes in the network—as opposed to local commit of the transaction on the leader. If desired, latency can be reduced by reducing batch size—albeit at the cost of transaction throughput.)*

It should be noted that throughput numbers depend on the Coco Framework and its optimizations, consensus and cryptographic primitives selected, and potentially other network governance policies. Microsoft expects to further increase throughput through framework optimizations for v1.0, with the caveat that some performance trade-offs may occur when additional security mechanisms are enabled.

Confidentiality

Microsoft worked with partner Mojix to port its supply chain DApp to a Coco-enabled Ethereum protocol. In general, DApps can be "lift and shifted" with minimal changes to work on top of an Ethereum ledger integrated with Coco; the only required changes are to support authentication to the Coco network and to leverage Coco's confidentiality features. As such, confidentiality can be implemented via access control policies and surfaced through ledger-specific constructs, without requiring complex cryptographic schemes.

Because Ethereum does not have a confidentiality model in Solidity (one Ethereum smart contract language) and because Microsoft did not want to modify the EVM, the prototype used TEEs to implement confidentiality through access control policies written within each smart contract. To do this, Microsoft disabled access to transaction and block level information in the Coco Ethereum adapter. Mojix developers were then able to enforce access control rules in the smart contracts themselves by verifying the address of the caller. All smart contract members were made private and all read functions were required to have a signature that was checked against a configured, authorized address list using *ecrecover()*.

For events, Microsoft added RPC methods that required a signature to check access before reading events. Mojix developers were then able to implement smart contract methods, with the Coco Ethereum adapter calling the smart contract function to check if the end user was authorized. If the check access method was absent, the smart contract was considered public within the Coco network.

To many, the confidentiality demo may appear to be a standard enterprise application. The beauty is in this fact; what appears to be a standard enterprise application backed by a centralized database is, in reality, an Ethereum blockchain application backed by a distributed ledger across organizations.

Viewing the Demos

A video recording of the two demos can be found here.

Conclusion

Blockchain is a transformational technology with the ability to drastically reduce the friction of doing business, especially in regard to shared business processes across organizations. A growing number of enterprises are investigating or experimenting with it as a secure and transparent way to digitally track the ownership of assets across trust boundaries, opening-up new opportunities for cross-organizational collaboration and imaginative new business models.

Microsoft is committed to bringing blockchain to the enterprise—and is working with customers, partners, and the blockchain community to continue advancing its enterprise readiness. Our mission is to help companies thrive in this new era of secure multi-party computation by delivering open, scalable platforms and services that any company—from ledger startups to governments, healthcare organizations, and global banks—can use to create new value.

The Coco Framework, our latest step in this journey, provides a foundation for blockchain in the enterprise by delivering the key elements that businesses are demanding. When a blockchain ledger implementation integrates with Coco, it approaches the performance and scalability of a traditional, centralized database. Coco also enables arbitrary data confidentiality rules for the contents of the ledger, such as role-based access control to transactional data. Finally, Coco delivers a comprehensive system for distributed governance rather than requiring outside management mechanisms.

As noted throughout this document, Coco is not a standalone blockchain protocol; rather, it provides a trusted foundation with which existing blockchain protocols such as Ethereum, Quorum, Corda, and Hyperledger Sawtooth can be integrated to deliver complete, enterprise-ready ledger solutions. By design, Coco is open and compatible with any blockchain protocol. It can even enable general-purpose, secure, and accountable multi-party computation outside of blockchain solutions.

J. P. Morgan (Quorum), R3 (Corda), and Intel (Hyperledger Sawtooth) have agreed to integrate their proprietary blockchain ledger platforms with Coco. Microsoft plans to launch the framework on GitHub in 2018 as an open source project—available on both Linux and Windows, in the cloud or on-premises. In the meantime, Microsoft will continue to work with customers, partners, and the blockchain technical and business communities to continue advancing production adoption.