Confidentiality in Private Blockchain

Stuart Popejoy <stuart@kadena.io>

Revision v1.0

August 2016

Abstract

Confidentiality in a blockchain setting refers to the ability for counterparties to a transaction to prevent other participants from knowing certain facts or details about the transaction. Confidentiality is seen as a critical feature for private blockchain adoption by industry, even as it threatens to undermine the very features that make blockchains attractive to industry, like perfect replication, unforgeable audit trails, and auto-reconciliation. This paper surveys approaches proposed and implemented for confidentiality in blockchain, evaluating their maturity, practicality, performance, and what kind of trade-off they might demand versus desirable blockchain features. We then detail confidentiality options offered by the Kadena blockchain, designed to strike the right balance demanded by the use-case between secrecy and functionality.

Privacy in Public Blockchain

The very notion of a "public blockchain" like Bitcoin or Ethereum seems to reject any idea of confidentiality: all transactions are "in the clear" allowing inspection of the entire ledger; addresses are anonymous but fixed, meaning individual payment activity is easily detected. Such transparency is of course by design, and is critical to maintaining trust in a cryptocurrency: every node verifies every transaction, preventing fraud; transparency even underwrites the probabilistic consensus model, as a hostile "fork" is impossible to hide. Nonetheless, measures can be taken to enhance anonymity, and methods have been proposed to hide key details of a transaction.

Enhancing Anonymity with KDFs

A widely-used technique for enhancing anonymity in a public chain is to use a fresh *key* (corresponding to an address) for each new transaction, managing a collection of keys as a *wallet.* While early wallets simply randomized keys, modern "deterministic wallets" generate keys using cryptographic *key definition functions* (KDFs) that reliably produce fresh keys from a "master" key, allowing all keys to be re-produced as needed from the master.

This cannot guarantee anonymity against behavior profiling or metadata correlation, requiring additional services and techniques like "tumbling" or mixing coins. Nonetheless a KDF-backed wallet is seen as sufficient to guard against opportunistic identification. We will return to KDFs in the context of on-chain encryption and forward secrecy, but we note here that KDFs can be seen as providing "just enough" privacy for certain scenarios.

Blockstream Confidential Transactions

Many solutions have been proposed to provide transaction confidentiality in public blockchain. Here, we consider Blockstream's *Confidential Transactions* (CT) which can be used to encrypt transaction amounts in Bitcoin. CT is a *commitment scheme*, that is, a way to allow parties to *commit* to values without exposing those values to other participants. In CT, a commitment has the additional property of supporting *homomorphic addition*, such that an

encrypted value can be “added” to another value without revealing the underlying amount.

CT’s additive commitments can slot directly into Bitcoin transaction amounts. Where a normal Bitcoin transaction verifies that the sum of inputs equals the sum of outputs, a CT transaction with all inputs and outputs expressed as commitments can similarly be summed and verified. While this does not hide the transaction’s existence itself or the keys used, transaction amounts are completely hidden, and can be revealed later by transactees as necessary.

Applicability to Private Blockchain

CT requires significant computational effort in the order of 1-5ms per transaction. In Bitcoin, the impact on verification performance is considered to be manageable, whereas a high-performance environment like Kadena would see huge decreases in throughput, so a cost-benefit analysis is required.

Most private blockchain use-cases have no need for “global ledger” a la Bitcoin. As a non-anonymous environment, transactions occur between authenticated counterparties, who can then use whatever trust model they mutually agree to. Entire transactions can thus be invisible to outside participants, while counterparties can “crack open” the transaction data to run smart contracts on fully-decrypted values.

In this environment, the value of CT commitments is diminished significantly. Commitment values would remain encrypted while the contract computes on decrypted values, creating “two worlds” of contract computation. This is not to say that commitment schemes have no value here – indeed later we will propose a far simpler approach – but instead that *computation on commitments* is not worth the expense or added complexity in these scenarios.

Confidentiality in Private Blockchain

In private blockchain, confidentiality is a core demand, so many solutions and proposals are in play. As we have noted, the authenticated environment allows for greater flexibility in approaches than in a public blockchain. We start

with the simplest approach, which is also our favorite: encrypting data on-chain.

On-Chain Encryption

It seems intuitive that if we want to use a blockchain, but we don't want to share information with everybody using it, we would simply encrypt data such that only intended participants can decrypt it. As noted above, this might prevent building a unified, self-reconciling environment like Bitcoin, but industry has clearly spoken: they neither need nor want that for most use-cases, preferring to cloak transactions in as much secrecy as possible.

The benefit of this approach is its simplicity, but also that it sacrifices none of the desired advantages of a blockchain: all data is still on the ledger, thus all participants still maintain an identical copy of the encrypted data, eliminating concerns over disaster recovery and replication, and providing drop-in *transactional infrastructure* that "just works."

Is Encryption Good Enough?

Some find fault with on-chain encryption as they do not believe it can ever be enough: if non-participants can see a blob of encrypted bytes pass by, that constitutes a security breach, or a regulatory violation.

For example, global payments use-cases have to abide by *data localization* regulations, restricting how data can enter or leave a jurisdiction, which is seen as preventing a global blockchain. While these issues are complex, specific restrictions around export of personally-identifiable information (PII) are easily implemented via smart-contract logic targeting those jurisdictions. Meanwhile, the goal of ensuring that the jurisdiction has a complete "local copy" of all constituent transactions is automatically afforded by a unified blockchain.

As for security concerns, we note that cloud companies like Amazon Web Services (AWS) have been successfully colocating competing businesses in the same datacenter (or computer) for years. If properly-managed encryption were not up to the job of securing each companies' deployments, AWS would be out of business. We assert that the rigorous use of secure encryption techniques like

forward secrecy and key rotation can fully satisfy cybersecurity and regulatory concerns.

Nevertheless, concerns about on-chain encryption have led many blockchain vendors to abandon encryption altogether. We consider these approaches now.

Centralization

Centralization refers to the practice of putting a blockchain in a central location, or privileging certain nodes of a blockchain cluster over others, to contain and localize data to prevent exposure.

In centralization, counterparties simply message transaction requests to these central locations, and receive tokens or some other crypto-hash-like "proof" that their transaction has gone through. Privacy is assured, as none of the participants have a full copy of the data, nor are they party to any external communication.

While this achieves the objective of confidentiality, the cost is too high. If a blockchain is a cryptographically-verified *distributed* ledger, centralization takes away distribution, and thereby eliminates a key motivation for blockchain altogether. Indeed, this approach offers no advantage over well-established utilities built on traditional database and messaging platforms.

We also reject the use of centralization as a technique to scale or speed up a blockchain. Kadena's unparalleled performance at scale eliminates the need for such workarounds.

Tokenization and Multi-Chain

Tokenization is a long-standing technique by which a blockchain-verified value is used to anchor some other transaction, and has been likened to taking out a classified ad in the New York Times with the transaction hash: its main function is to uniquely identify and record the transaction in some verifiable system of record. An infinitesimal Bitcoin transfer can tokenize any transaction in this way, powering the "first wave" of industry blockchain adoption.

Clearly, tokenization is no longer necessary with smart contracts on a performant and scalable private blockchain: we can directly model our solution on-chain. However, tokenization remains as a way to link multiple chains together as a *multi-chain*. Again, this is sometimes used to provide increased performance; we simply note that standard *sharding* or "striping" approaches (assuming the data permits such partitioning) can accomplish similar scaling feats with a blockchain like Kadena.

Multi-chain can be used as a confidentiality solution, where separate blockchains are formed for each combination of counterparties, preventing transmission of transaction data to foreign chains. This is woefully complex from an operational perspective, which is usually addressed by some form of centralization. Again, we see a key advantage lost, here the radical operational simplicity of a single blockchain.

Other Approaches

Zero-knowledge proof is a commitment scheme used in protocols like Zerocash, which attempt to produce a perfectly anonymous and confidential cryptocurrency system. These are still experimental and quite slow, with latency in the minutes, and are also focused on a *total ledger* which is not useful for many private chain use-cases.

Verifiable Secret Sharing is another scheme used in *secure multiparty communication* systems like Enigma. These provide *decentralized private computation* like lotteries or elections and thus have limited application here; we mention them only for completeness, and because Enigma and others *make use of* a blockchain as a permanent record and authorization engine.

Confidentiality in the Kadena Blockchain

To understand the options for confidentiality in Kadena, we start with a trivial option that can be supported by public blockchains as well: *externally-encrypted payloads.*

Externally-Encrypted Payloads

Bitcoin, Ethereum and Kadena can all support the ability to store "extra data" with a transaction. This data payload can of course be encrypted in such a way that only transactees are able to decrypt it upon receipt, producing a form of confidentiality that is quite useful. For instance, a simple, confidential (if not anonymous) "signing workflow" interaction, like signing a legal contract on-chain, can be accommodated with encrypted payloads alone. However, smart contracts are unable to compute on this data, as it enters and exits the blockchain in encrypted form.

Confidential Smart-Contract Execution

To enable confidential smart contract computation, we need a way to associate the execution environment with a cryptographic identity.

Entity/Node Association

In a public blockchain, all nodes are functionally identical. In a "permissioned" blockchain, cluster nodes have distinct cryptographic, meaning every node is "known" to the cluster. However, these consensus identities are strictly external to the transactional data maintained by the blockchain. Still, these fixed node identities make it possible to associate specific nodes with the business entity that "owns" and operates them, enabling a form of *automatic authentication,* where a node can operate on behalf of the business entity without user interaction.

This opens the door to smart-contract confidentiality, as a message can be encrypted such that only particular *entities* can decrypt it. Now, any message that an entity can decrypt can be offered to the smart contract engine automatically for execution.

Disjoint Databases

All blockchains maintain a *log* of transaction data. Incremental hashes, consensus enforcement, and public-key cryptography are used to ensure that this log is identically replicated to every node in the cluster. In Bitcoin, this log contains the inputs and outputs of "accepted transactions," but in *deterministic smart*

contract blockchains like Ethereum and Kadena, this log contains inputs to the smart contract engine only. Once fed into a smart contract, these inputs produce outputs that bear little or no resemblance to the log data. We refer to this dataset generated by the smart contracts as the *database*.

If none of the inputs are encrypted or otherwise hidden, this database will emerge identically on every node in the cluster. Once we have distinct business entities, however, we see that different entity nodes will necessarily process different inputs. As a result, the databases are *disjoint* between the different entities.

Disjoint Databases and On-Chain Encryption

Disjoint databases represent a new operational scenario for blockchains, and inform our preference for *on-chain encryption.* An alternative approach to confidentiality is to tokenize transaction hashes and distribute inputs directly to entity nodes for smart-contract execution via secure side-channels. However, the disjoint databases produced can no longer be reproduced from the blockchain inputs alone, requiring dedicated replication and disaster recovery.

By keeping the encrypted data on-chain, we maintain the operational benefits of a blockchain. Bringing up new entity nodes is easily achieved, and the blockchain ensures that the entity's distinct database is replicated.

On-Chain Encryption

We return to our main approach to confidentiality, which is to symmetrically encrypt transaction data onto the ledger that can be automatically decrypted by node entities participating in the transaction for execution in the smart-contract engine. The operational simplicity and elegance of this approach is too attractive, and the alternatives too costly, to consider any other approach.

Symmetric encryption is susceptible to breach by hostile parties, for which we utilize proven techniques to offer *forward secrecy* and support *key rotation.*

Forward Secrecy through Diffie-Hellman and KDFs

Clearly, entity-node association involves the node producing the necessary keys to decrypt transactions. However, the naïve use of these "static" keys to encrypt data is dangerous, as a breach of encrypted data on the ledger would lead to all entity transactions being revealed.

Guarding against this requires using distinct keys for one or more transactions, under the heading of *forward secrecy*. For this we employ state-of-the-art techniques to produce a Diffie-Hellman shared secret to seed the generation of keys with KDFs, and can recover these keys as needed, much like a Bitcoin deterministic wallet.

Key Rotation

Key rotation is a central need at all levels of a private blockchain. Consensus keys must be able to be securely transitioned for security purposes and to support membership changes. The Pact smart-contract language natively supports key rotation to allow management of bespoke key-based authorization schemes for protecting data access.

For confidentiality, key rotation is critical to respond to an in-house security breach (as opposed to the ledger data being cracked, which is covered by forward secrecy), so that counterparties can securely receive new public keys representing the business entity. Regular defensive key rotation is a best practice in any case.

Simple Commitments

With a high-performance, confidential blockchain solution like Kadena, organizations can rest easy knowing their data is safe from exposure but still perfectly replicated. Within this scenario, there is still room to support simple commitment schemes, and even CT, although the latter requires special support in the smart contract layer to perform the homomorphic addition.

A transactee can simply commit to a value by encrypting it in the transaction payload such that only that entity can decrypt it. While other parties cannot verify its contents, the commitment can nonetheless be revealed in a dispute resolution process easily. For instance, a simple payment transfer could

commit to the sender's current balance as a promise that the "check will not bounce". As most inter-organizational private blockchain deployments will involve some kind of regulatory framework, participants can be compelled to provide valid commitments with each transactions and decrypt them on demand.

Conclusion

We have presented a survey of confidentiality techniques in use in private and public blockchains, before detailing the Kadena solution, supporting confidential smart-contract execution with properly managed on-chain encryption and simple commitments, maintaining all the robustness and operational advantages of a single blockchain.