A.i. BlockChain
Artificial Intelligence

KafkaBlockchain Library

KafkaBlockchain is a java library for tamper-evidence using Kafka. Messages are optionally encrypted and hashed sequentially. The library methods are called within a Kafka application's message producer code to wrap messages, and called within the application's consumer code to unwrap messages. A sample utility program is provided that consumes and verifies a blockchained topic.

Because blockchains must be strictly sequentially ordered, Kafka blockchain topics must either have a single partition, or consumers for each partition must cooperate to sequence the records. Sample programs demonstrate blockchains with a single partition and with multiple partitions. If multiple producers exist, they must cooperate to serially add records to a Kafka blockchain.

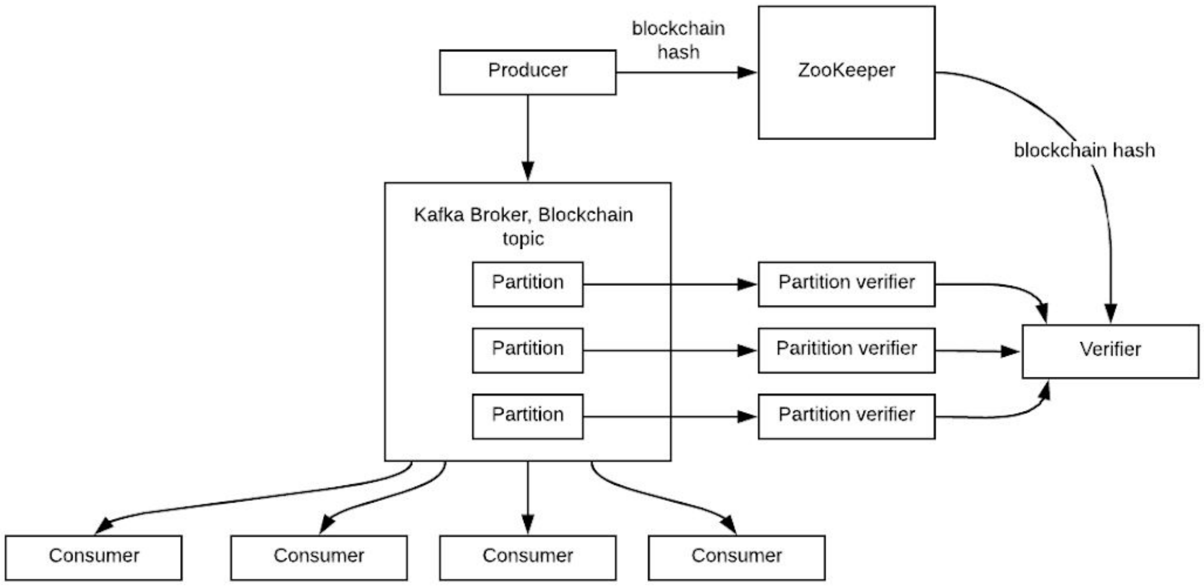
Kafka already implements checksums for message streams to detect data loss. However, an attacker can provide false records that have correct checksums. Cryptographic hashes such as the standard SHA-256 algorithm are very difficult to falsify, which makes them ideal for tamper-evidence despite being a bit more computation than checksums.

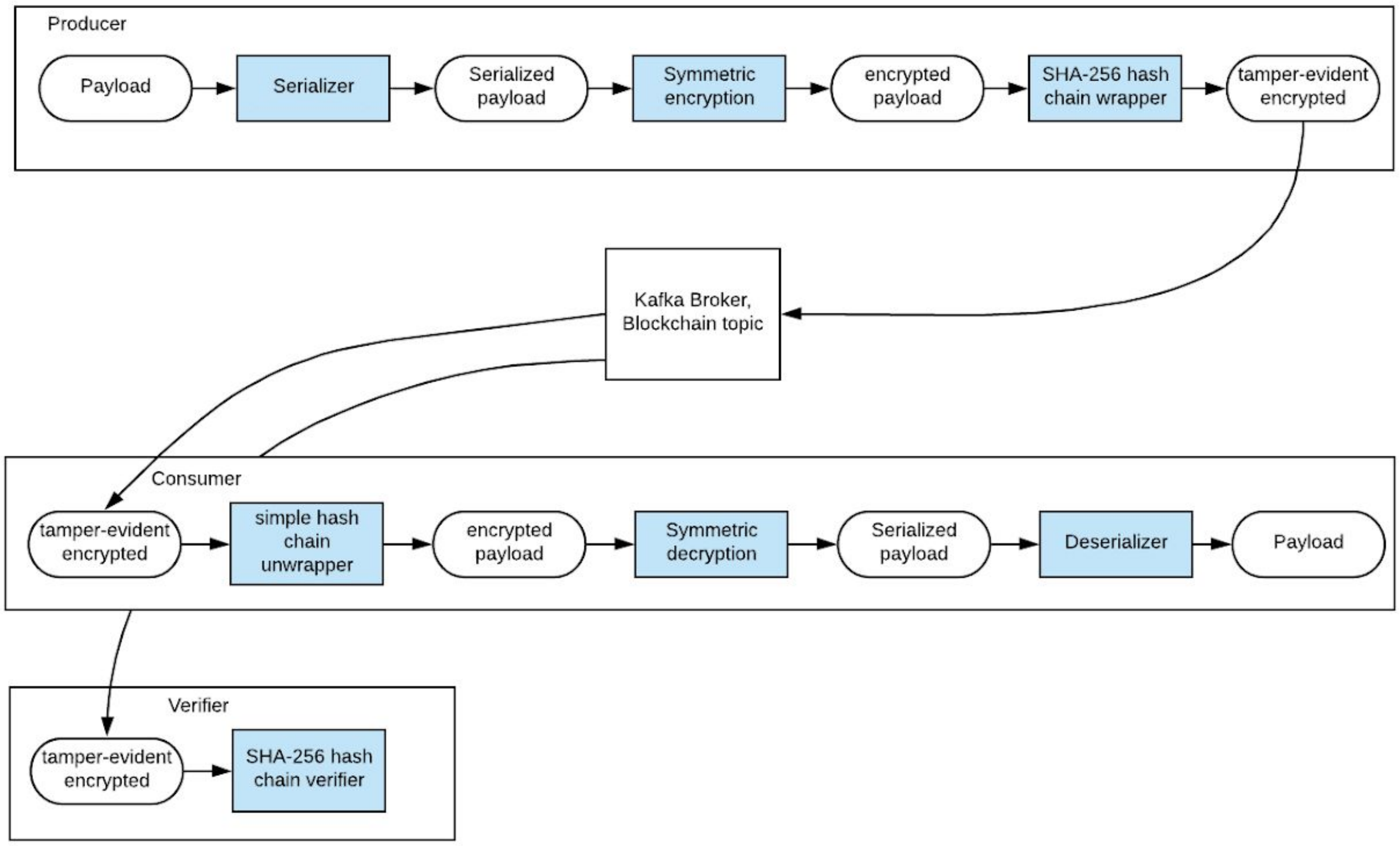
To manage Kafka blockchains, the sample programs store the first (genesis) message SHA-256 hash for each blockchain topic in ZooKeeper. In production, secret-keeping facilities, for example Vault can be used. Likewise a sample program demonstrating encryption as well as tamper-evidence can be adapted to store its symmetric encryption key in either ZooKeeper or a secret-keeping facility.

Dependencies This library uses the Bouncy Castle crypto library. Apache Maven is required to build this library, and to run the quickstart examples. This library is written using Java 10.

Github: https://github.com/ai-coin/KafkaBlockchain

KafkaBlockchain





As shown in the diagram above, the KafkaBlockchain library methods are embedded in both the application producer and consumer functions. The application payload data is deserialized to bytes and optionally encrypted with the Bouncy Castle library using the symmetric method which is relatively fast. The encrypted payload is then assigned a sequence number and wrapped in a tamper-evident Java object which calculates a SHA-256 hash over the encrypted payload and the remembered hash of the preceding tamper-evident record. Thus a hash chain is created in which the most recent record in the chain contains hash information over all the preceding records. The tamper-evident Java object is sent to the Kafka broker for recording and distribution.

The reverse process is performed by the application record consumers. The record is retrieved from the blockchain topic, then extracted simply from the tamper-evident Java object. The encrypted payload is then decrypted using the private key. The decrypted bytes are serialized back into the application payload data.

KafkaBlockchain verification uses a verifying consumer thread for each partition to reassemble the records in the order in which they were produced.

The sample programs illustrate a SHA-256 blockchained topic with sequential records issued from a single producer with multiple Kafka partitions. The library can be scaled to multiple producers by increasing the complexity of the verifier threads to accommodate the separate hash chains that are sequentially created by each producer. The enhanced verifier can ensure that each producer’s records are free of tampering and falsification. The enhancement consists of tracking each producer’s records and the expected sequence number that is stored in the Java wrapper around the application’s data payload.