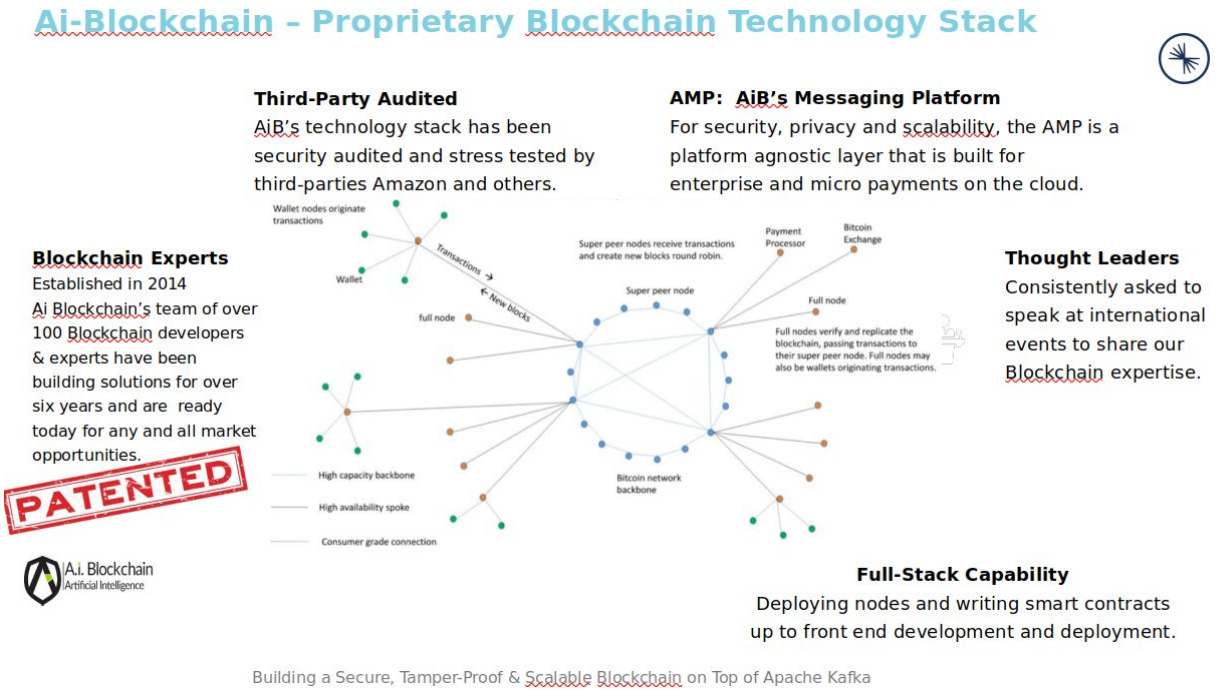
A.i. Blockchain
Artificial Intelligence

Confluent Webinar Script for July 14, 2020



Slide: Ai-Blockchain – Proprietary Blockchain Technology Stack

This diagram shows the scalable hub-and-spoke AiB token blockchain network.

The center ring consists of super peer nodes in which intelligent software agents record token transactions issued by the peripheral nodes.

Every 10 minutes, a new tamper-proof block to token blockchain which is compatible with the proven Bitcoin protocol.

Ai-Blockchain patented the method whereby network consensus is achieved by having the super peer nodes take turns creating new blocks without costly proof-of-work.

The patent improves ordinary token blockchains by achieving immediate transaction settlement using standard servers and very efficient processing.

The key to the patent is that the super peers are managed by intelligent software agents.

The agents check each other's work and ensure that each super peer has a non-corrupted identical token blockchain.

The software agents themselves cannot be tampered with because their logs and messages are made tamper-proof.

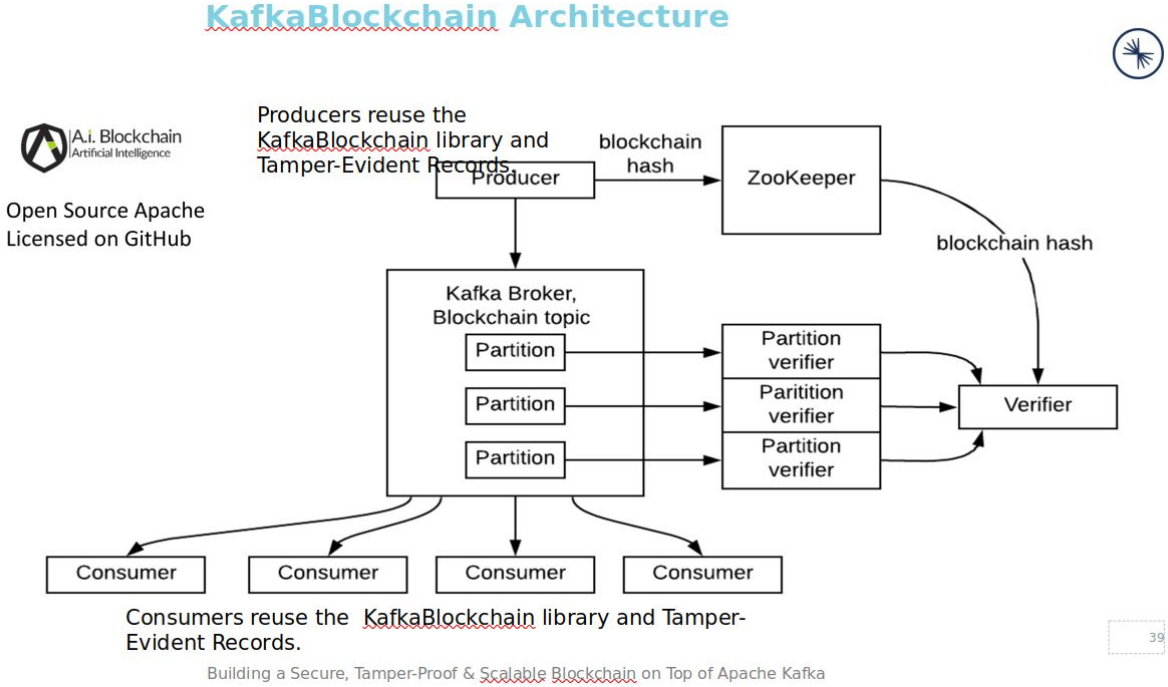
This is where Apache Kafka comes into action.

Every log file and message stream is an Apache Kafka topic.

Ai-Blockchain's open source Java library, named KafkaBlockchain, makes the agent logs and message streams tamper-proof.

An attacker cannot change any of the token blockchain data, nor any of the software agent logs, without detection.

Let's take a closer look at how the KafkaBlockchain Java library works.



Slide: KafkaBlockchain Architecture

This diagram depicts a Kafka broker interacting with a record producer and four record consumers.

The Producer reuses the KafkaBlockchain library to wrap payloads into tamper-proof objects.

Each Consumer reverses the steps, using the library, to unwrap payloads from tamper-proof objects.

Optionally, the KafkaBlockchain library encrypts payloads for the Producer, and likewise decrypts payloads for the Consumer.

The sequential records in a blockchain topic are made tamper proof by the KafkaBlockchain library using the well-known SHA-256 cryptographic algorithm.

It is practically impossible to falsify a given payload that yields the expected 256 byte precomputed SHA-256 hash value.

ZooKeeper is used to maintain the current blockchain SHA-256 hash value for each blockchain topic.

A simple KafkaBlockchain has a single topic partition, but this illustration shows three topic partitions.

More parallelism can be achieved in Kafka with multiple partitions when the application permits.

The blockchain verification process can be separated from the ordinary reading of the topic.

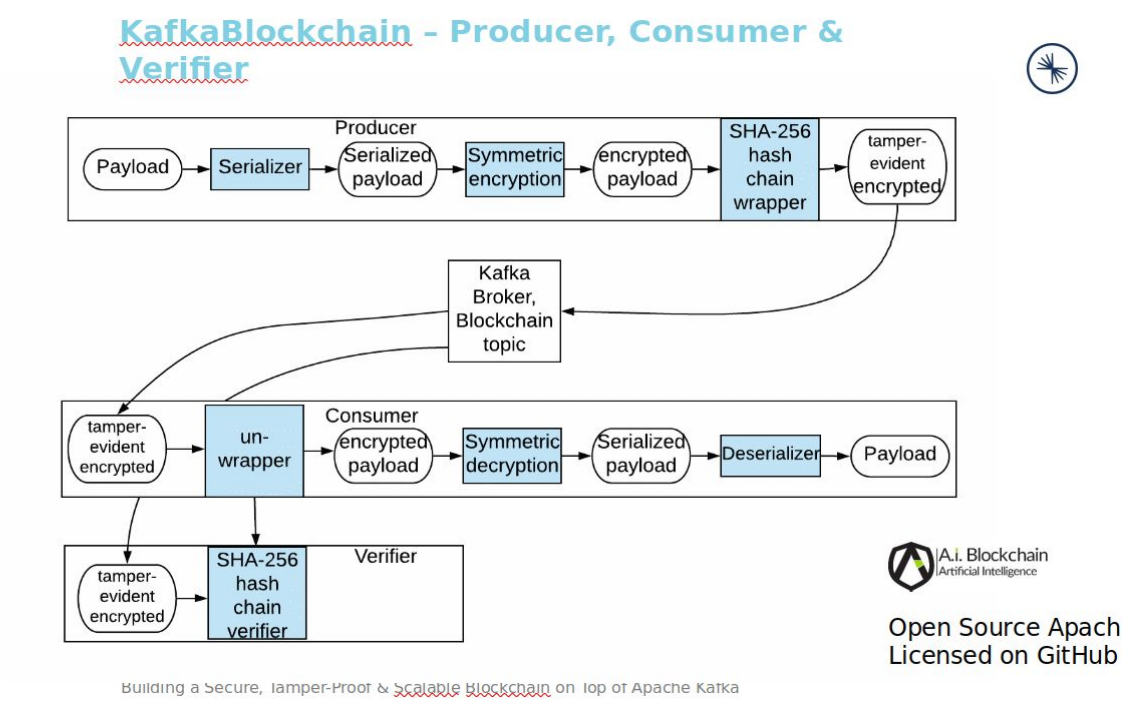
Starting at the first, genesis, record of the topic, a special Consumer can reuse library methods to easily recompute and verify the SHA-256 hashes for each record in sequence.

The blockchain topic records must be verified in sequence because the SHA-256 hash value of the previous record is an input into the computed hash of the current record.

Consequently, a corrupted bit in any of the preceding records changes the calculated SHA-256 hash of the current record.

That is how a blockchain works for tamper-proofing, each record is dependent on its predecessor in the sequential record chain.

Let’s now look into the processing steps of the tamper proof Producer, Consumer and Verifier.



Slide: KafkaBlockchain - Producer, Consumer & Verifier

The Producer is the Kafka component that puts new records into a given topic.

The Consumer is the Kafka component that gets records from a topic.

The Verifier is the special consumer that verifies that the topic has not been corrupted.

Here are the two or three steps of each KafkaBlockchain producer.

> A Kafka payload is typically a serialized Java object.
>
> Java serialization can be greatly speeded up by customized objects as demonstrated in the provided sample code.
>
> The Producer first step serializes the Java object into a byte array which becomes the serialized payload.
>
> The application use case may require encryption and the Producer optional next step encrypts the serialized payload bytes using an efficient symmetric cryptographic algorithm.
>
> The key for encrypting/decrypting a particular blockchain topic is stored in ZooKeeper, or in a production secret-keeping vault.

The next Producer step is always performed, and it wraps the serialized payload bytes, the previous record's SHA-256 hash, and the next serial number into an immutable tamper-evident record.

A field in this tamper-proof record contains the SHA-256 hash of the fields in this record.

If there are multiple producers for this particular Kafka blockchain, then they must cooperate to submit the records in order by serial number.

Here are the two or three steps of each KafkaBlockchain Consumer.

The tamper-proof record is polled from the blockchain topic.

The record's serialized payload field is optionally decrypted.

The payload Java object is obtained by deserializing the serialized payload.

If there are multiple consumers for this particular Kafka blockchain, then they must cooperate if it's necessary to process the records in strict order.

Here are the steps for the KafkaBlockchain Verifier.

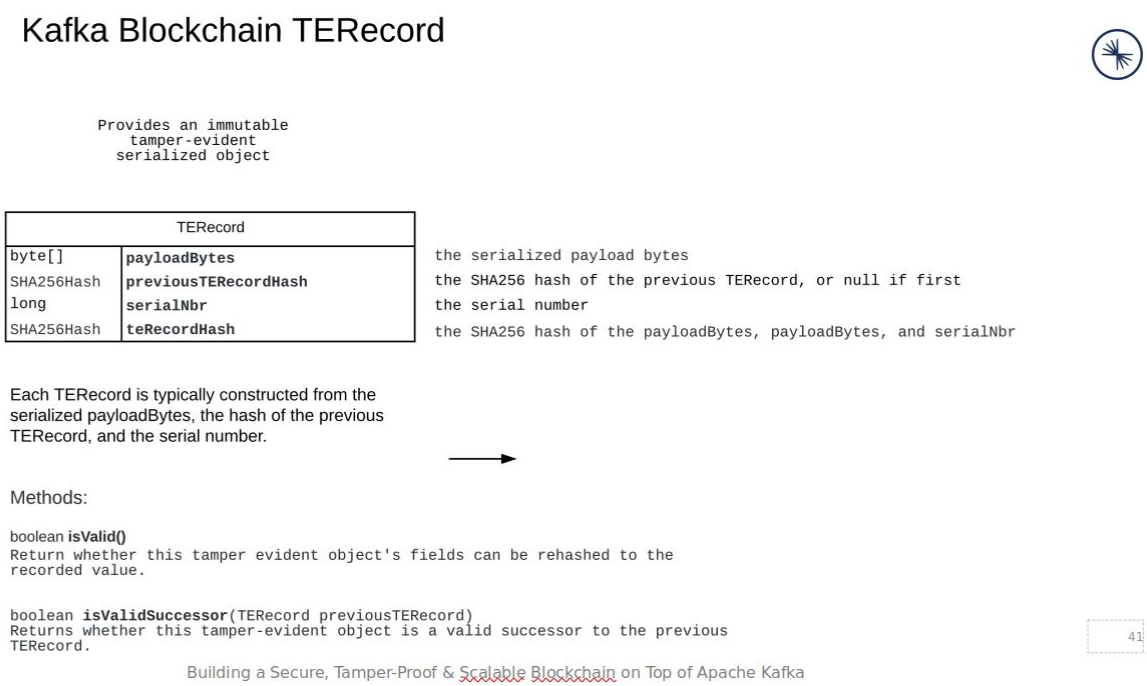
The next sequential tamper-proof record is polled from the blockchain topic.

If there are multiple partitions in the topic, a Verifier is needed for each partition - and they must cooperate to recompute the records in order by serial number.

The record is verified by recomputing its SHA-256 hash, and by ensuring that the value of the previous record's SHA-256 hash actually matches the recomputed hash value of that previous record.

The entire blockchain is verified by verifying each of the records from the first to the last in sequential order by serial number.

Let's now look at the Java Tamper Evident record provided by the KafkaBlockchain library.



Slide: Kafka Blockchain TERecord

This provides an immutable tamper-evident serialized object ready to put into a Kafka topic.

There are only four fields.

The first is the serialized payload bytes.

It is obtained by serializing the given Java payload object.

The sample programs use the built in Java serializer, with the sample payload object implementing the Externalizable interface for maximum performance.

The second field is the SHA-256 hash of the previous TERecord, by serial number, in the topic.

When a producer creates TERecords, it needs access to either the previous TERecord, or its SHA-256 hash.

When a verifier audits TERecords, it needs to start at the first one and verify them all in serial number sequence until the blockchain topic is exhausted.

The verifier ensures that the final SHA-256 hash matches the persisted SHA-256 for the blockchain, which is updated by a Producer every time a TERecord is created for the Kafka blockchain topic.

The third field is the serial number of the tamper-evident record, indexed from 1.

A single producer simply increments the previous value to populate this field.

Multiple producers must cooperate to obtain the previous SHA-256 hash, and to increment the serial number without conflict.

The final field is the SHA-256 hash of the payloadBytes, the previousTERecordHash and the serial number.

The KafkaBlockchain library provides several convenient constructors for the TERecord, typically given the payload Java object, the SHA-256 hash value of the previous TERecord in the blockchain, and the serial number.

The library provides two essential methods.

The first one is a boolean method named isValid with no arguments that returns whether this TERecord is valid.

It is true only if the recomputed SHA-256 hash matches the value stored in the record.

The second method is named isValidSuccessor with the single argument being the previous TERecord.

It is true only if this record is valid and its previous TERecord SHA-256 field matches the teRecordHash field of the previous record object.