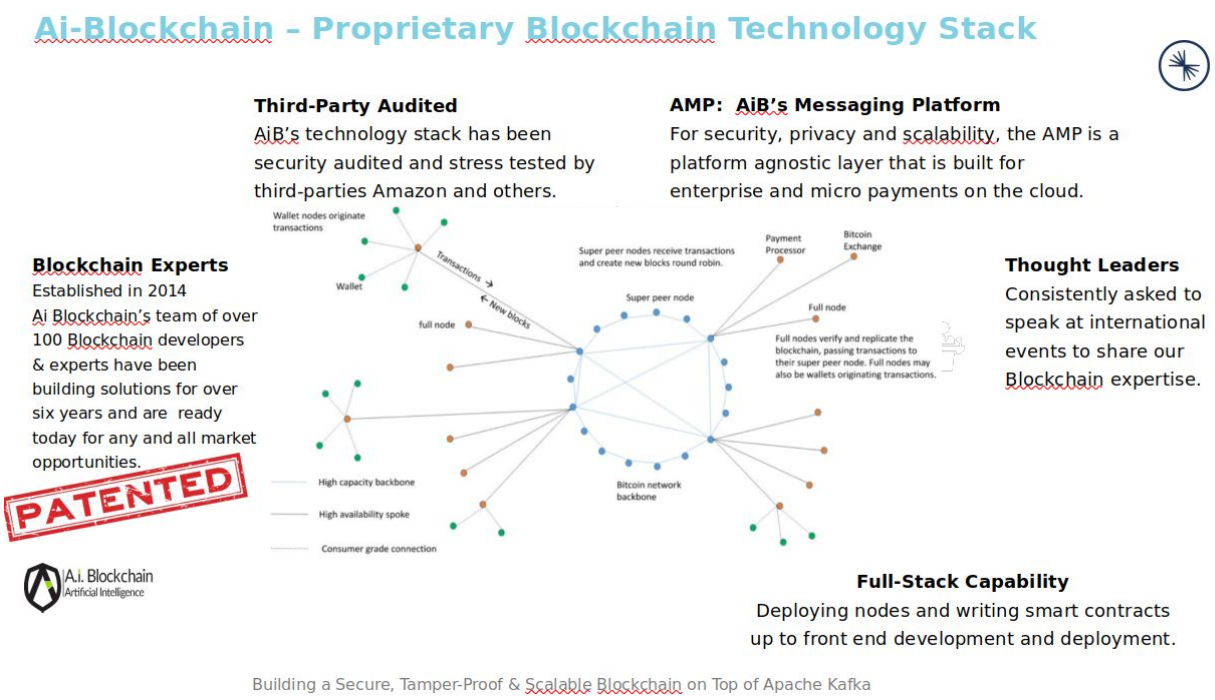
Confluent Webinar Script for July 14, 2020

Timings 11:30 minutes, 8:56 minutes, 9:10 minutes

Thank you Kai.

I am Stephen Reed, chief Scientist for Ai-Blockchain, and developer of the open source Kafka Blockchain library.

Lets take a look at the Ai-Blockchain technology stack and how Kafka and blockchain make it happen.



Slide: Ai-Blockchain – Proprietary Blockchain Technology Stack

This diagram shows Ai-Blockchain's scalable hub-and-spoke Bitcoin compatible 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, new token transactions are recorded in a 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.

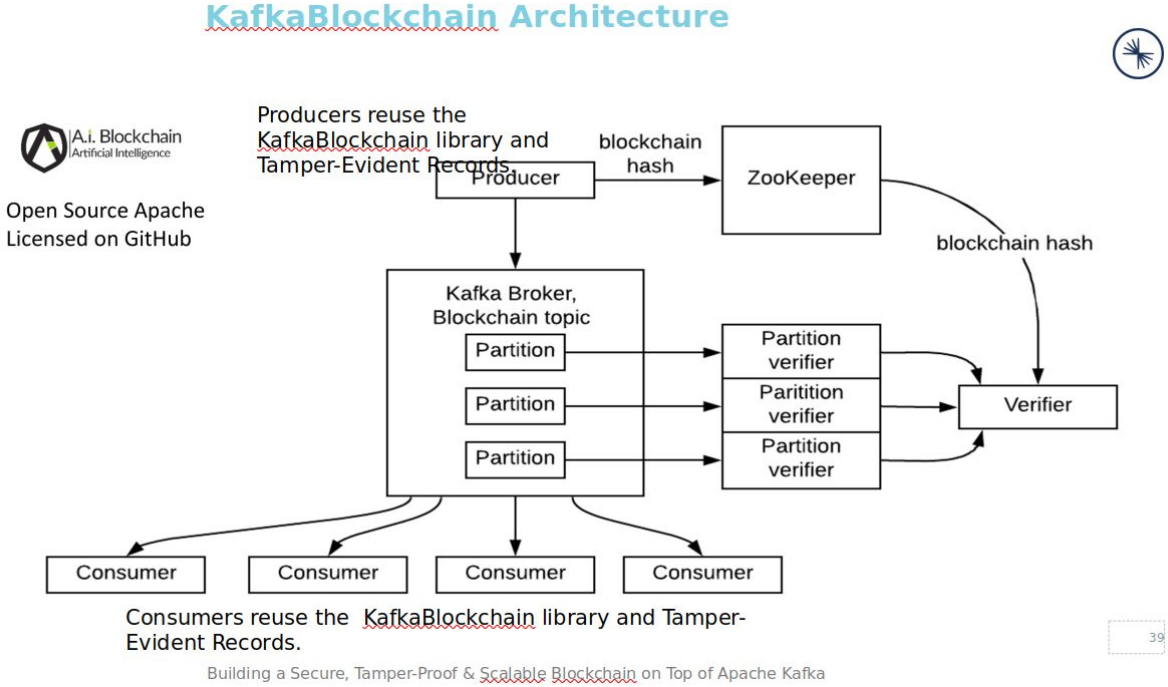
This is where Apache Kafka comes into action.

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

Ai-Blockchain's open source Java library, named Kafka Blockchain, 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 Kafka Blockchain Java library works.



Slide: Kafka Blockchain Architecture

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

The Producer reuses the Kafka Blockchain library to wrap payloads ***into*** tamper-proof objects which are ***sent*** to a blockchain topic.

Each Consumer reverses the steps, using the library, to unwrap payloads ***from*** tamper-proof objects which are ***received*** from a blockchain topic.

Optionally, the Kafka Blockchain 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 Kafka Blockchain library using the well-known SHA two fifty six cryptographic algorithm.

This algorithm inputs any digital data, crunching it all down into a particular two hundred fifty six byte value.

This computer hashing algorithm has the desirable property that it is practically impossible to falsify a given payload such that its hash matches the original's hash.

Any digital object is made tamper-proof by first computing its cryptographic hash and storing that value.

Subsequently the digital object can be verified by recomputing its cryptographic hash, and then comparing that result with the stored value for that particular digital object.

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

A simple Kafka Blockchain 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.

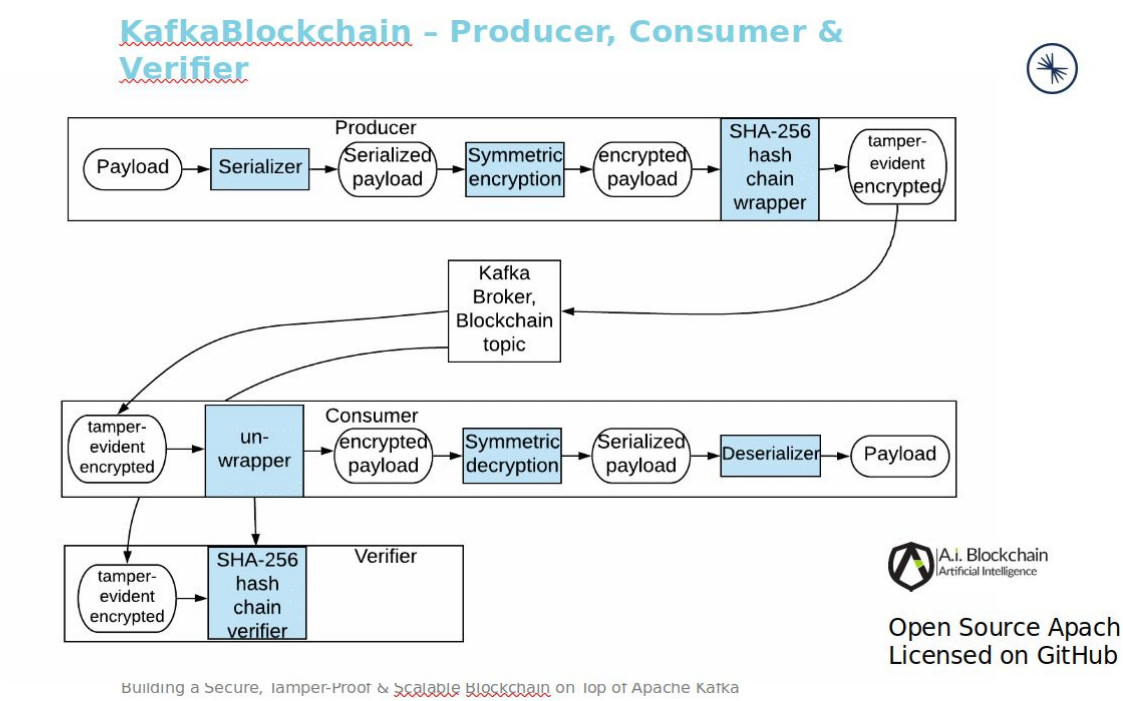
Starting at the firs**t, genesis**, record of the topic, a special Consumer can reuse library methods to easily recompute and verify the cryptographic hashes for each record in sequence.

The blockchain topic records must be verified in sequence because the cryptographic 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 cryptographic 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: Kafka Blockchain - 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 Kafka Blockchain 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 first step serializes the Java object into a byte array which becomes the serialized payload.
>
> The application use case may require encryption and the 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 step is always performed, and it wraps the serialized payload bytes, the previous record's cryptographic hash, and the next serial number into an immutable ta**mper-evident record.**

A field in this tamper-proof record contains the cryptographic hash of the other 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 Kafka Blockchain 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 Kafka Blockchain Verifier.

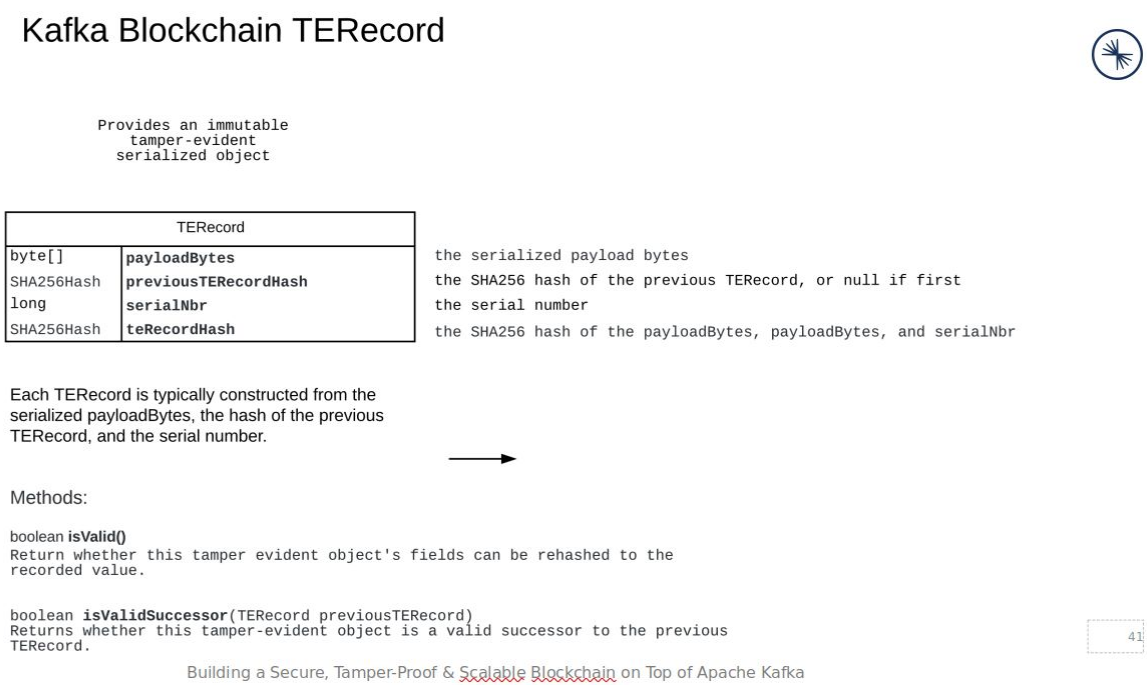
The next sequential tamper-proof record is polled from the blockchain topic.

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 cryptographic hash, and by ensuring that the value of the previous record's cryptographic 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 Kafka Blockchain library.



Slide: Kafka Blockchain tamper-evident record

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 cryptographic hash of the previous tamper-evident record, by serial number, in the topic.

When a producer creates tamper-evident records, it needs access to either the previous tamper-evident record, or its cryptographic hash.

When a verifier audits tamper-evident records, 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 cryptographic hash matches the persisted cryptographic hash for the blockchain, which is updated by a Producer every time a tamper-evident record 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 cryptographic hash, and to increment the serial number without conflict.

The final field is the cryptographic hash of the payloadBytes, the previous tamper-evident recordHash and the serial number.

The Kafka Blockchain library provides several convenient constructors for the tamper-evident record, typically given the payload Java object, the cryptographic hash value of the previous tamper-evident record 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 tamper-evident record is valid.

It is true only if the recomputed cryptographic hash matches the value stored in the record.

The second method is named isValidSuccessor with its single argument being the previous tamper-evident record.

It returns true only if this record is valid and its previous tamper-evident record cryptographic hash field matches the tamper-evident recordHash field of the previous record object.