IOTA

"A fabric for securely connecting and operating the Internet of Everything"

Author: Andre Szykier, IOTA Holdings LLC.

CONTENTS

INTERNET OF THINGS APPLICATION FABRIC

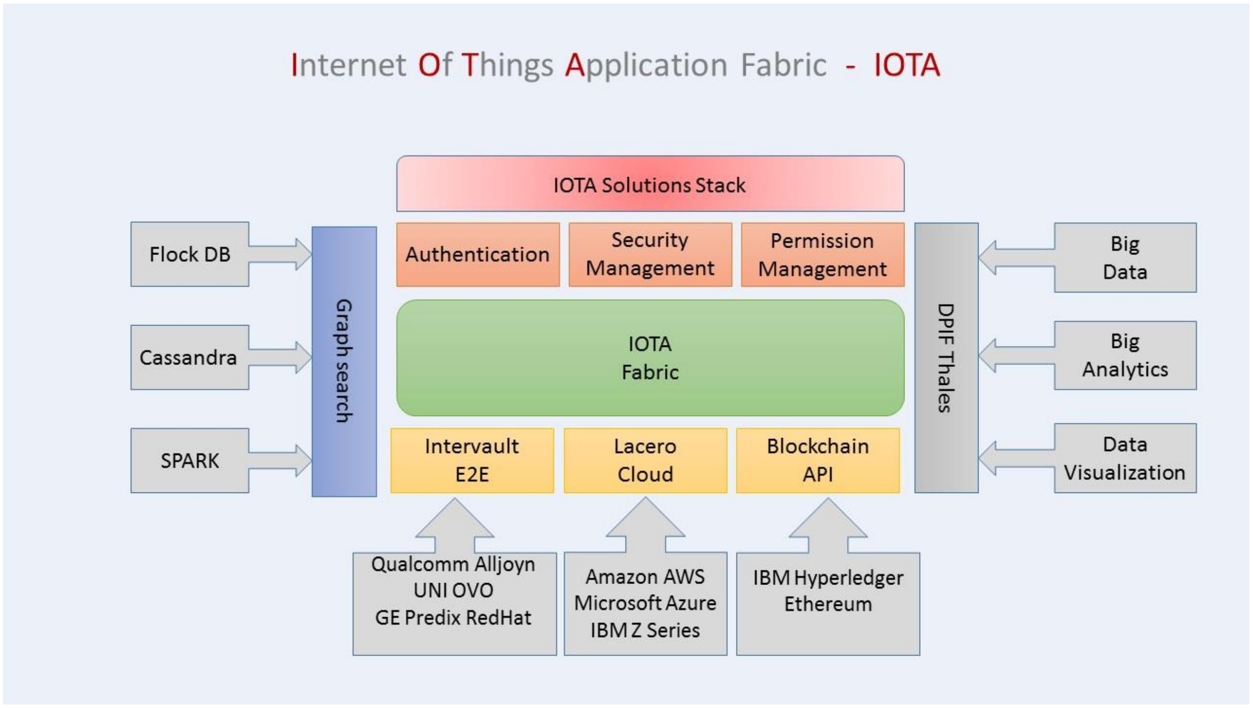
IOTA Holdings, a Delaware Company, was formed by senior specialists in the world of applied cybersecurity. Its mission is to implement advanced tools for Internet of the future – Peer to Peer.

IOTA recognizes that the current architecture of centralized web services, cloud computing and data security are deficient when operating in a P2P using IPV6 addressable landscapes. We also predict that the current environment supporting machine to machine (M2M) applications will not be adequate to handle secure P2P services across private as well as public cloud and device centric scenarios.

The goal is to implement the many decades of our work in military, business, medical and government projects in creating a *Fabric* rather than a *Platform* approach to addressing the unique challenges that the Internet of Things (IOT) and fast forwarding to the appearance of the Internet of Everything (IOE).

"A recent DDOS attack to traditional websites (IPV4) harnessed hundreds of thousands of unprotected IOT devices that brought down servers that translate web URL to the IP addresses, not once but several times in a single day. The attack crippled most of the major Websites that relied on these domain name gateways. This is the future of the next Internet. A battle of IPV4 vs IPV6 is underway."

WHAT IS IOTA?



IOTA Fabric has several components that connect services securely without regard to their purpose. Different components come in to play depending on need. Some are focused on protecting the devices/services that connect to identical or compliant devices/services. Others protect data in motion while others secure data at rest. Authentication of devices and services is an emerging problem, in first use and in subsequent operation. Enabling end to end (E2E) encryption that is unencumbered by issuing and protecting digital certificates is necessary to enable P2P transactions. As business employs mobile transactions using smart pay applications, an emerging issue is how to authenticate and protect the transaction details (metadata) in a distributed, non-centralized fashion. As financial transactions migrate to non-banking centralized systems (Bitcoin), integrity and audit are paramount.

IOTA Fabric addresses these issues by bringing together various computer-based methods together, for the first time. IOTA collectively addresses security through its Fabric to:

- Discover, register and authenticate devices and services on first use
- Provide protocol gateways for connected devices in heterogenous networks
- Deliver E2E security for any transaction (event) with session based keys
- Extract data from silo mainframe storage into secure cloud storage
- Secure temporary or permanent data storage in hybrid public or private clouds
- Enable businesses to deploy private block chains for their own environments
- Record transactions using block chain methods from participating vendors
- Connect devices and services using graph search algorithms
- Monitor all transactions without the need for a centralized database.
- Permit real time deactivation of any IPV6 addressed service or device.
- Allow decertification of IPV6 addresses (culling) for business related decisions.
- Meet or exceed US DISA requirements for secure remote access.
-

IOTA OPERATES ON 4 BASIC PRINCIPLES:

- The ability to bring a device or service into the IOTA fabric
- The commitment to protect and deliver data without compromise
- The right of participants based on permissions to query any transaction
- The protection of distributed P2P transactions from alteration or deletion

What does this mean?

First, any business or government agency that wants to take advantage of IOTA Fabric (*IOTA^{F}*) needs only to register a service once that supports IPV6 addressing.

For example, a smartphone (most already support IPV6) can use its IP address as a means for registering any application that runs on the device. In such a scenario, the device IPV6 can register any downloaded application.

However, the application provider must register itself once with its own IPV6 address to *IOTA^F*. The device links to other devices using its IPV6 private of public software defined network (SDN) and the group of IPV6 applications onboard.

If it is a device independent service (a browser enabled transaction, for example), the service provider must register and assign an IPV6 address to any of its subscribers. The management of browser based events are outside the scope of this document but many solutions can be implemented. Once authenticated, *IOTA^F* manages whether that address is permitted to commit to a transaction and if necessary, create a record in a block chain event.

Second, *IOTA^F* is all about protecting and delivering data, secure and unaltered. We do not offer or recommend proprietary secure methods or algorithms to clients. We adhere to standard encryption methodologies that do not require certificate based handshakes between one or more IPV6 addresses. Our solution, *InterVault*, creates session keys or varying strength to provision an encrypted channel through which third party data is sent. If the data is unmangled (layman term for encrypted) it will remain unchanged at its destination. The same for encrypted data. The pipe protects data regardless in whatever form content was created. Our methods are industry standard compliant.

IOTA^F also is committed to protecting data at rest. This is an increasing problem for cloud and virtual application providers who outsource their infrastructure to IT vendors. Without deprecating any such provider, security only goes far as the EULA contract with their clients. If a penetration of data occurs, most outcomes fall on the side of negligence on the part of the client. To this end, *IOTA^F* gives users control in a way that even if the IT cloud provider environment is compromised by anyone, the data is useless. The technology is very mature, and enables users to know that data can be recovered, without modification at any time. The component called *Lacero*, shreds, replicates and mangles source data across cloud as well as distributed devices while permitting full recovery, whether a device, storage unit or virtual service is online, offline or malfunctioning.

Third, permission rights are an important part of any transaction. In current webcentric applications, this right is usually in the favor of the service provider, be it a bank, retailer, a telecom carrier, a web enabled service or variants. Scroll down the EULA and users find that all roads to curing a bad event point in favor of the provider. This has more import than just transaction integrity; all actions that relate to the event, including everything about the user through permissions can benefit the provider.

In a P2P Internet, both parties have rights and tools such as block chains and encryption to level the playing field. But how are permissions verified, audited and concisely expressed? By providing a granular permission resource that exists and maintains a historical record that is undeniable by either party. Not at the service agreement level alone. At the transaction level. This modality creates a more flexible approach to data exchange. Data can exist for the time of a transaction and disappear once completed but a hash of its metadata persists in a block chain.

Metadata can be hashed and stored in a block chain but the details preserved in a distributed, redundant, fragmented data record, within the cloud or across known devices. Permission management services are embedded and customizable by *IOTA^F* enabled services or device users.

Fourth, P2P operates on different principles than users of web centric services. Multiple factor authentication (MFA) and single sign on login (SSO) are not readily suited for this network model. Many researchers are examining the block chain method for quick authentication, even key management and distribution. *IOTA^F* embraces this approach as a viable alternative.

HOW DEVICES/SERVICES CONNECT IN A DISPARATE IOT UNIVERSE

Projected forecasts of IPV6 addressable devices are numerous. At least 50 billion by 2020 and hundreds of billions more by 2025. Home automation, sensors, appliances, webcams, security monitors and so on can now connect through the web through their device address or unique device signature. The biggest growth however, will be in services, software code performing one or more steps of an event where information is the output. No one can reliably forecast this volume as some of these services may create transient, rather than permanent transactions.

Consider an electrical utility using smart meters or sensors managing and recording power production and consumption. Meters are connected locally with data over power circuits but in an ad hoc, nearest neighbor mode. Only the meter closest to a base station transfers data for all local meters to a central place or receives commands to operate devices in a home that talk through their own smart meter. This store and forward data has a short half-life but how is such a system protected from third party interference?

Today it is ripe for malware but with *IOTA^F* the periodic hash of every meter read can be used to validate that the energy metadata is correct, creating a block chain event. Likewise, with ad hoc mesh networks relying on Wi-Fi, radio spectrum or dedicate hard connections. In a *fabric* design the objective is to be able to recognize, connect and monitor devices and services that exist in their own ecosystems. For example, devices that operate with the ZigBee protocol require a gateway through APIs to connect to other protocols. Here are just a few in the IOT world:

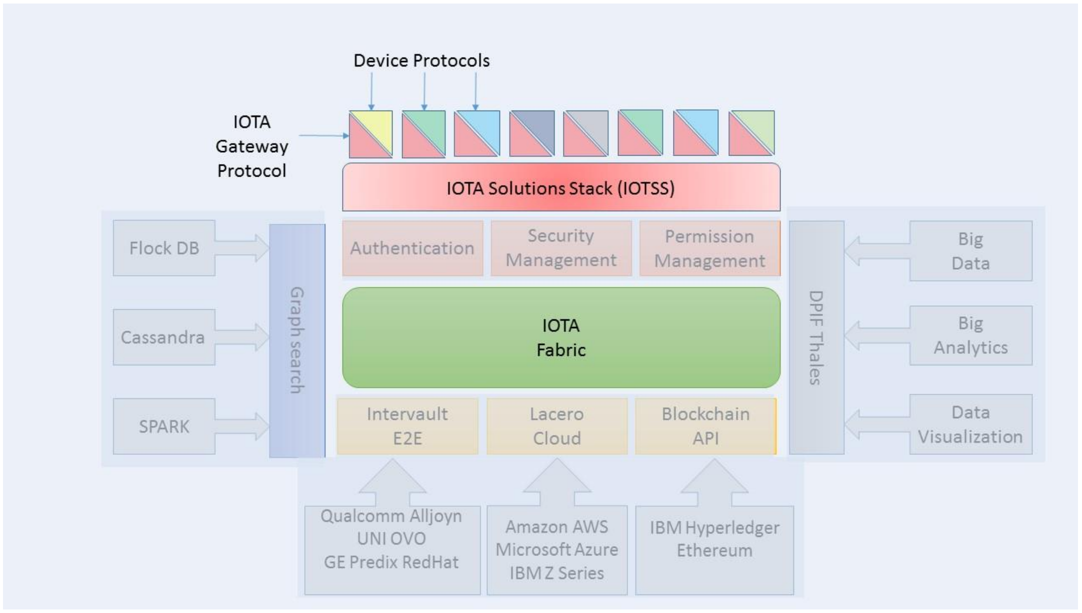
1. Infrastructure (6LowPAN, IPv4/IPv6, RPL)
2. Identification (EPC, uCode, IPv6, URIs)
3. Communication / Transport (ex: Wifi, Bluetooth, LPWAN)
4. Discovery (Physical Web, mDNS, DNS-SD)
5. Data Protocols (MQTT, CoAP, AMQP, Websocket, Node)
6. Device Management (TR-069, OMA-DM)
7. Semantic (JSON-LD, Web Thing Model)
8. Multi-layer Frameworks (Alljoyn, IoTivity, Weave, Homekit)
9. Infrastructure (ex: 6LowPAN, IPv4/IPv6, RPL)
10. Identification (ex: EPC, uCode, IPv6, URIs)

11. Comms / Transport (ex: Wifi, Bluetooth, LPWAN)
12. Discovery (ex: Physical Web, mDNS, DNS-SD)
13. Data Protocols (ex: MQTT, CoAP, AMQP, Websocket, Node)
14. Device Management (ex: TR-069, OMA-DM)
15. Semantic (ex: JSON-LD, Web Thing Model)
16. Multi-layer Frameworks (ex: Alljoyn, IoTivity, Weave, Homekit)

As you can surmise, most protocols organically grew based on a vendor design and were not intended to talk to unknown devices, only compatible ones. It would be foolish to think that one standard can service many different types, using a 'boil the ocean' approach. It is possible to design a gateway protocol that encapsulates the metadata and data of any system such that it securely transports content between heterogenous protocols. Medical devices are one example. A pacemaker device, an insulin pump, an edema monitor, all carry different protocols for connectivity as well as unrelated payloads. What is needed is a translation table for any devices that maintains,

1. Data metrics unique to the device/service treated as an object
2. Metadata on the type of registered device/service
3. Standard specifications to common attributes such as address, timestamp, etc.
4. Communication protocols that permit transmission across various scenarios

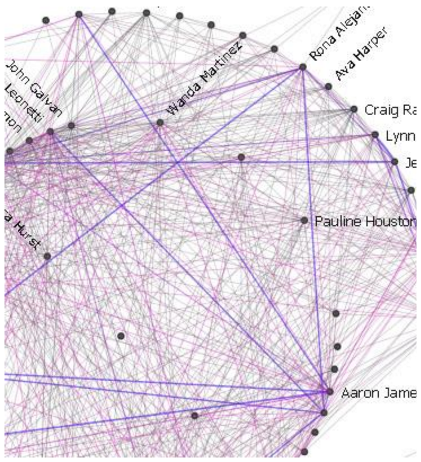
This is not as difficult as it seems. The W3C consortium has promoted standard definitions across a wide range of data types, attributes, translators, alternative definitions and so on. *IOTA*^{F} is committed to supporting the W3C methodology in its protocol gateway design and exposes this to the open source community for improvement and additions.



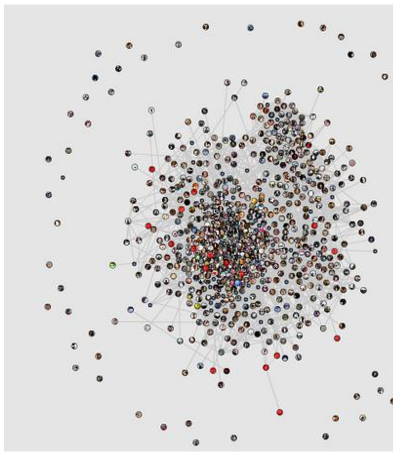
This is accomplished in the IOTA Solutions Stack (IOTSS). A registered device has a known protocol connected through the IOTSS API to the IOTA Fabric protocol. Another device with a different protocol can also connect using a socket and the Fabric allows data as objects and metadata as class attributes to be exchanged between connected devices/services. This is analogous to a router in a software defined network (SDN.)

DISCOVERING DEVICES OR SERVICES THROUGH IOTA

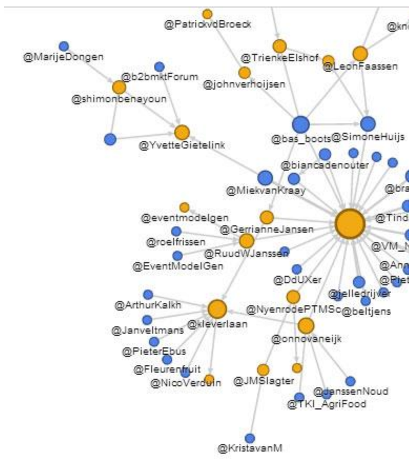
One of the challenges in social media was the ability to connect and maintain connections between users, as friends in a private space and to outsiders in the public space. People who follow people who follow other people quickly devolves into a stochastic, constantly changing, network. The picture below shows 3 examples of this phenomenon for Facebook, Google+ and Twitter.



Facebook



Google+



Twitter

These examples demonstrate the various types of networks that exist in the human space. **Facebook** gravitates to a broader edge network with people having multiple links, many due to sharing timelines within people they allow to connect with. Occasionally a post or comment will spill over into the public commons where Facebook decides how content is shared using various algorithms. **Google+** is more like joining what they refer to as 'circles' where both friendship but more about common interests dominate the graph. Those that contribute little but are in a circle are in the periphery of the graph. Contributors and shares dominate the center.

Twitter is a more interesting topology in that its prime reason is for people to share thoughts albeit in compressed text, a sort of virtual shout-out. Those who become interesting attract followers whose tweets are funneled into a larger group of followers. This network has nodes that are magnets where a follower can be in more than one group. The magnets in and of themselves do not create larger magnets but spawn more followers because traffic in one is exposed across multiple nodes. The difficulty in managing these links caused Twitter to scrap their earlier designs because they could not scale nor keep the social graph dynamic enough because of the nature of their shallow linking. Here interests are spawned by events causing a flood of tweets for short periods rather than moving at the pace of a social network like Google+ or LinkedIn. To solve their problem, they designed **FlockDB**.

Written in Scala, FlockDB is an open source distributed, fault-tolerant graph database for managing wide but shallow network graphs (not to be confused with neural network layers) to store relationships between users, e.g. following and high frequency posters. It differs from other graph databases that are optimized for multi-hop graph traversal but rather aims at rapid execution of messages. In its GitHub release it depends on **Gizzard**.

Gizzard is an open source sharding framework to create custom fault-tolerant, distributed databases. It operates as a middleware networking service running on JVM. It manages partitioning data across arbitrary backend data stores for fast access. The partitioning rules are stored in a forwarding table that maps key ranges to partitions. Each partition manages its own replication through a declarative replication tree. Gizzard handles both physical and logical shards. Physical shards point to a physical database backend whereas logical shards are trees of other shards. Note that *sharding*, in this methodology is more about spreading records or partitioned datasets across multiple processors, not storage devices. It does not shard the data objects themselves.

FlockDB is a distributed graph database for enumerating relationships in a many to many format. Twitter best describes its advantages as:

- a high rate of add/update/remove operations
- potentially complex set arithmetic queries
- paging through query result sets containing millions of entries
- ability to "archive" and later restore archived edges
- horizontal scaling including replication
- online data migration
- multi-hop queries (or graph-walking queries)
- automatic shard migrations

FlockDB is much simpler than other graph databases such as neo4j because it tries to solve fewer problems. It scales horizontally and is designed for on-line, low-latency, high throughput environments such as web-sites. Twitter uses FlockDB to store social graphs (who follows whom, who blocks whom) and secondary indices. As of April 2010, the Twitter FlockDB cluster stores 13+ billion edges and sustains peak traffic of 20k writes/second and 100k reads/second.

How does this relate to the world of IOE? Surprisingly a lot. The difference lies in the initial discovery stage among devices and or services. In the IOT data space there are many different topologies. Here are some examples.

Smart meters are basic polled devices. Once they are centrally registered by the utility to a physical location, there is little discovery needed. They collect energy readings per some unit of time, and either send periodically to the nearest meter for forwarding or are polled through the same pathway by the central system. Sometimes they are connected to home automation devices that can be controlled by the utility. HVAC interruption is a typical case. Signals can be sent as data over power or through radio or Wi-Fi channels. In the latter case, IP addressing comes into play.

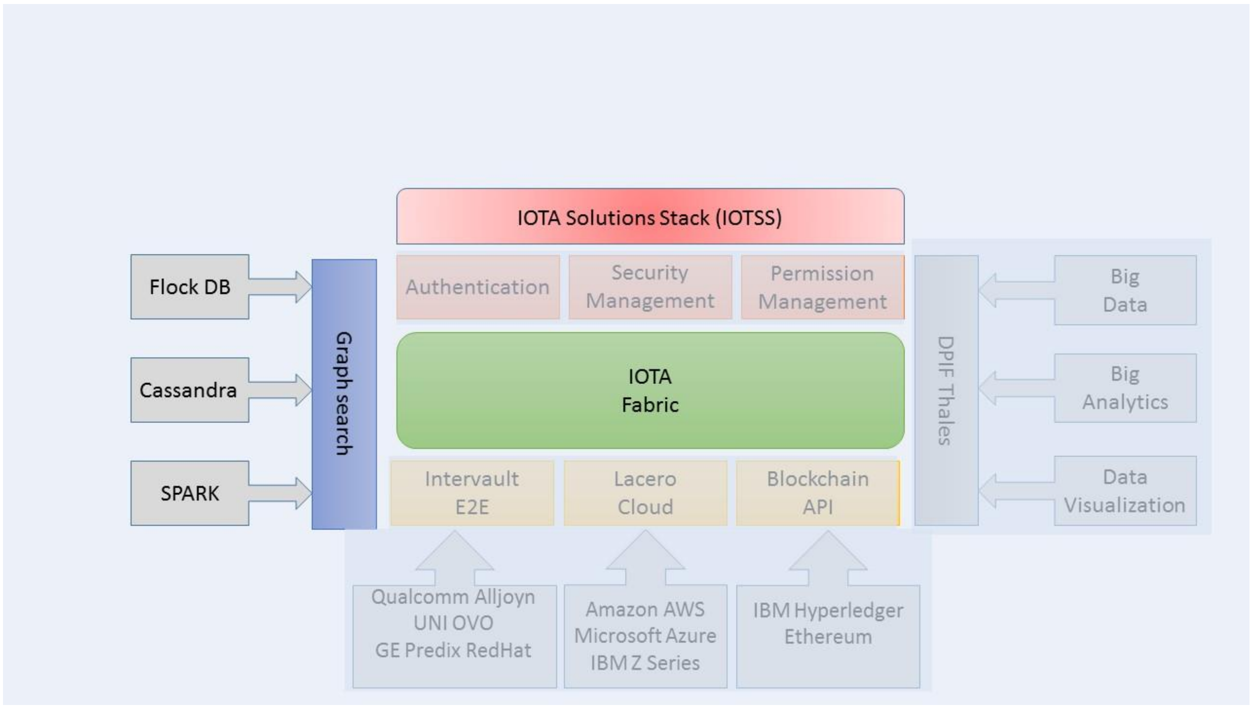
A **medical device** such as an edema monitor that checks on several conditions of a recovering patient at home is more complex. It acts more as an alert system to a central service point that can call for paramedic assistance. Here the device pushes data readings out on a schedule. However, a remote medical system can also spawn readings not to a central point but to another system, talking to multiple points of analysis. EEG, pulse oximeter readings glucose levels, movement sensors, heart rate monitors can all go separately to different collectors that can perform appropriate responses or forward to a central point if triage is necessary.

An appliance (smartphone, computer, PlayStation or Xbox) operating under **BitTorrent** network is where data objects (usually video), are spread around enough users, as a complete object or a segment. BitTorrent protocol allows users to join a "swarm" of hosts to upload to/download from each other simultaneously. The protocol is an alternative to the older single source, multiple mirror sources technique for distributing data. Knowing which hosts can service a request requires a central database of host/object tables. Rather than streaming from a single source, the protocol permits the assembling of partial objects from multiple sources thus reducing bandwidth. How hosts are discovered depends on their purpose. For streaming content deemed protected by copyright, much of this traffic is relayed through proxy servers which mask the origin and destination IP addresses through multi-hop relays.

> A distributed data object is divided into segments (data shards). As a peer receives a new shard it becomes a source for other peers to access. Each shard has a cryptographic hash so that any modification can be detected from accidental and malicious modifications.

IOTA^F uses the graph search method to allow application designers of its Fabric to take advantage of various components to support IOT/IOE services. A client network can be private (like a web service subscriber model), public (gateway to connect disparate networks, or hybrid (connecting private and public networks in a transaction-based event). In the last case, the role of block chains can play an important role.

The graph search algorithms are based on FlockDB in that the IOTSS protocol can broadcast metadata attributes out through devices, either directed by a database address, either by proximity as defined by the application (nearest neighbor or master node) or through random walks based on IPV6 addresses (very slow).



IOTA^F conforms to the NoSQL model for mapping attributes across very large data sets. As such it supports query schemas supported by Cassandra and Apache Spark data stores. The FlockDB algorithms can process query patterns against networked devices to find like, similar, exact or partial matches.

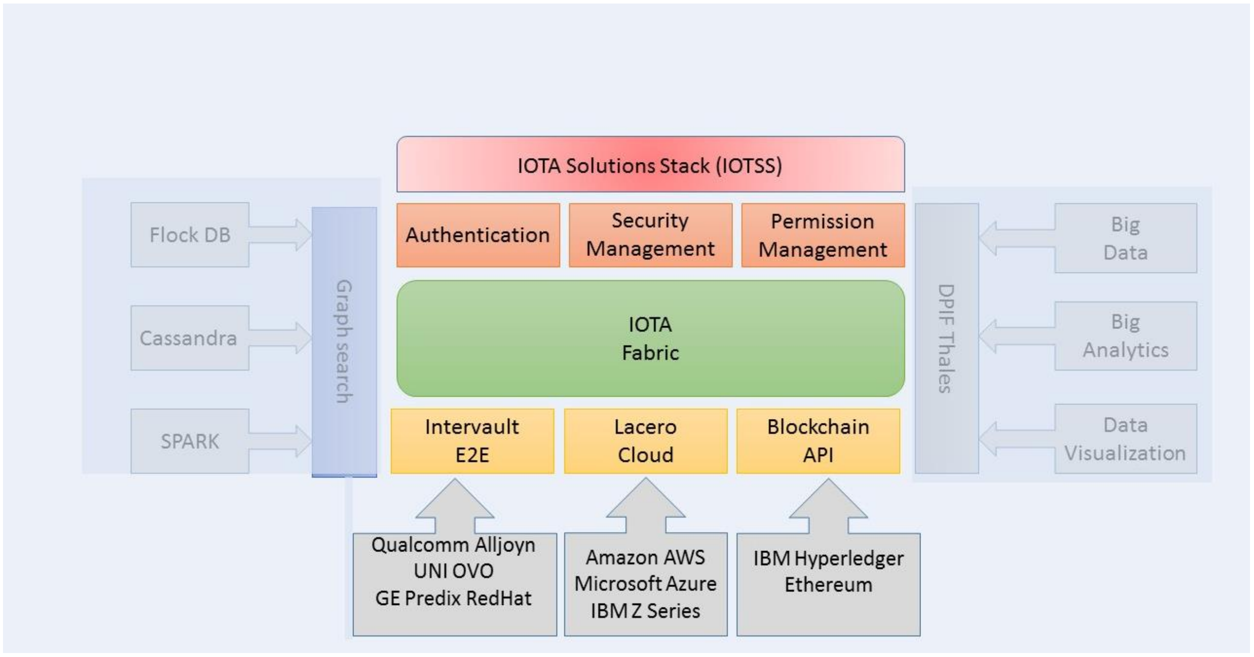
INTERVAULT – LACERO – BLOCKCHAIN

A block chain is a like a public ledger but with encrypted copies distributed among nodes in a network. The main premise is there is no need for a central database. An entry in one copy is distributed to all other node block chain entries.

Attempting to change a ledger entry using a blockchain network is not impossible but difficult. You must alter the *entire* ledger on 51 per cent of all network nodes. If the ledgers are not *simultaneously* altered, the nodes detect the entry as illegal and prevent the change. What makes it even more difficult is that blockchains are both encrypted and hashed, each entry or group of entries chained to the previous segment in the block.

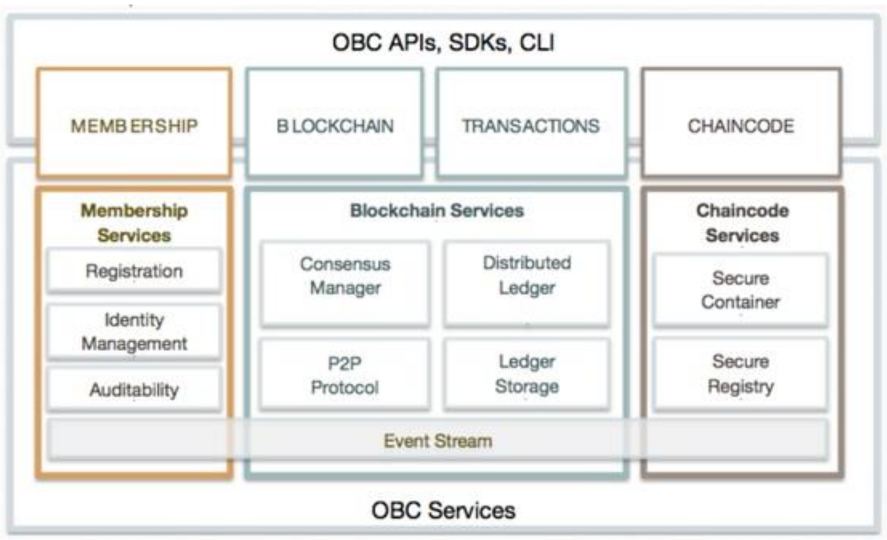
So, what is the main benefit using blockchains? On the blockchain, all transactions contain metadata with a unique entry identifier which acts as a proxy for the 'owner' making them private. Access to an entry is governed by permissions that may use single, multi-factor authentication along with digital signatures that are unique to a device or a service. Because no central command exists, nobody has the authority or access to facilitate reading and by design, changing an entry. The methods used, and there are many variants to a blockchain structure, are too detailed to provide here. A starting review and technical guide is provided here.

IOTA^{F} through its IOTSS enables different blockchain architectures using SOAP and JSON methods the exchange metadata, identifiers and data object(s) between one or more IPV6 addressable devices, one or more being a node in a blockchain enabled network.



The Authentication, Security Management and Permission Management modules establish E2E secure connections between device/service(s) and the blockchain node(s), expose classes of cryptographic methods available for use and interrogate permission filters for allowing or blocking connections.

The best example of how *IOTA*^{F} connects with a blockchain process is to examine IBM's open source Hyperledger, initially designed for distributed financial transactions. It follows the Open BlockChain architecture below.



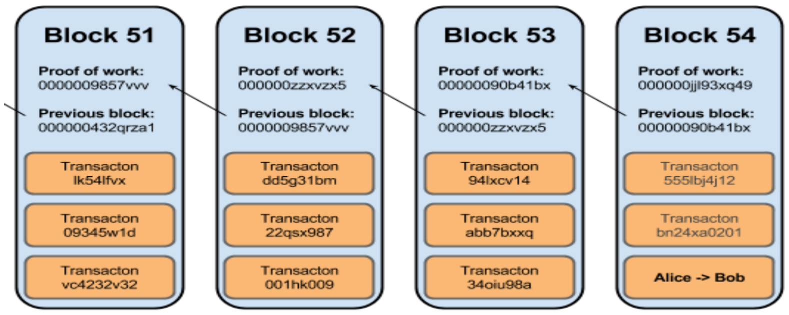
IBM's distributed ledger protocol is run on *nodes*, or as they refer to as *peers*. It distinguishes between two kinds of peers:

A *validating* node on the network is responsible for running consensus, validating transactions, and maintaining a copy of a ledger.

A *non-validating* node functions as a proxy to connect clients (issuing transactions) to validating peers. Non-validating peers do not execute transactions but may verify them.

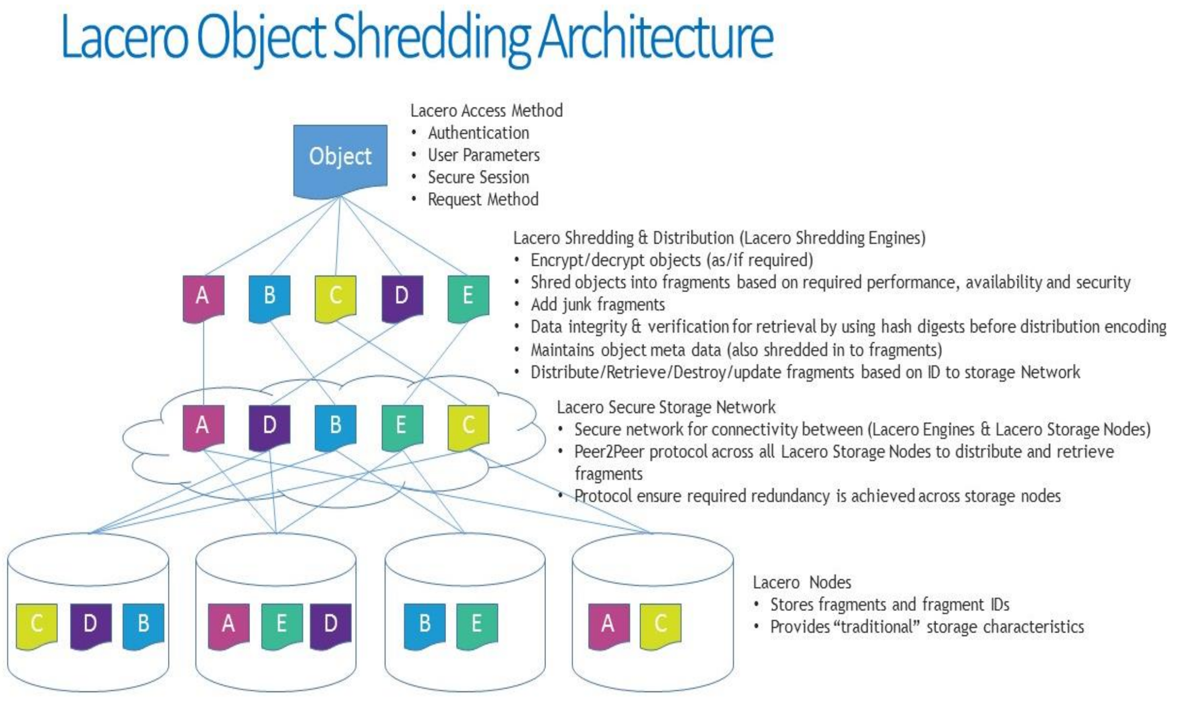
Validating peers execute a consensus protocol in a replicated state machine that accepts three types of transactions as operations:

- *Deploy* transaction: Takes a chaincode (representing a smart contract) as a parameter; the chaincode is installed on the peers and ready to be invoked.
- *Invoke* transaction: Invokes a transaction of a chaincode installed earlier through a deploy transaction with application specific arguments. The chaincode executes the transaction, may do read and write entries, and indicates whether success or failure.
- *Query* transaction: Returns an entry of the state directly from reading the peer's persistent state. This is part of the audit of the network.



The *IOTA^F^* **Blockchain API** does not provide its own blockchain but acts as a gateway for devices or services that require a distributed ledger network access. However, two other components (InterVault and Lacero) extend a chosen blockchain network by providing secure, distributed, redundant storage in the cloud or across a federated cluster of devices with storage capacity.

Lacero is a next generation virtual data store that can run on public, private, hybrid cloud storage systems. In a sharding process, data objects are fragmented one or more times, each with different fragments and distributed among storage devices. The storage devices use the Authentication module to connect to a sending or receiving device. Multiple storage devices are first mapped to each other in a multicast way. E2E encryption using public/private key exchange occurs through the InterVault module. Storage devices are then addressable in a many to many network.


Lacero Object Shredding Architecture
Object
Lacero Access Method
• Authentication
• User Parameters
• Secure Session
• Request Method
Lacero Shredding & Distribution (Lacero Shredding Engines)
• Encrypt/decrypt objects (as/if required)
• Shred objects into fragments based on required performance, availability and security
• Add junk fragments
• Data integrity & verification for retrieval by using hash digests before distribution encoding
• Maintains object meta data (also shredded in to fragments)
• Distribute/Retrieve/Destroy/update fragments based on ID to storage Network
Lacero Secure Storage Network
• Secure network for connectivity between (Lacero Engines & Lacero Storage Nodes)
• Peer2Peer protocol across all Lacero Storage Nodes to distribute and retrieve fragments
• Protocol ensure required redundancy is achieved across storage nodes
Lacero Nodes
• Stores fragments and fragment IDs
• Provides "traditional" storage characteristics
A B C D E
A D B E C
C D B
A E D
B E
A C

The object is passed to a sharding node, either in the client environment or to a virtual server in a cloud environment. Each fragment is distributed to one or more storage nodes which may or may not keep the node depending on a status register.

The first node removes the status and may or may not keep the fragment. Usually it passes on the fragment to one or more other nodes. Through a randomization process at least one node will accept the fragment and increment that status register and perhaps pass it on to other nodes. Storage nodes that store the fragment update the status register until a threshold is met at which time no further nodes are used and the fragment id with a keep status flag is returned to the sharding nodes.

The process is repeated until all fragments from all object copies are known to be distributed across storage nodes. Then each sharding node, creates a metadata object of fragment ids, order of assembly and other proprietary parameters that allow an object to be reconstructed from its distributed fragments. The metadata can optionally be fragmented as well and stored across the same or different network nodes. Typically the object metadata id is stored in a private storage area while the metadata fragments and object fragments can exist in a public cloud environment.

The principles for redundant distributed storage go back 27 years. In 1999 Carnegie Mellon University Computing Lab designed **PASIS** and the first implementation by the author occurred in 2001 as ***Chromosome***. Since that time further improvements were made to allow the method to work with clustered infrastructure and cloud computing environments. The key differentiation is as following:

Sharding nodes create metadata to reconstruct fragments of an object stored on one or more storage nodes but have no knowledge which storage nodes or data partitions the fragment resides.

- Storage nodes can determine by a fragment id if it is stored on its node but not which object the fragment belongs to.
- Both metadata and fragment data are hashed to prevent tampering. These hashes can also be stored separately. Neither sharding nodes or storage nodes retain the object metadata. This is returned to the requestor, again through secure channels. It is the responsibility of the requestor to store the metadata in a secure manner.
- All communication between nodes has E2E encryption generating session keys that are arbitrarily reset by the Authentication module to prevent man in the middle attacks.

The Authentication module stores a digest of storage nodes identifiers that indicate which were made available for an object, class or group or type of objects. This digest does not indicate the storage nodes that stored fragments but identifies nodes that were made available. This digest is passed as part of the metadata. It acts as a list to the Authentication module when requesting the connection to multiple storage nodes (temporary private network) to retrieve an object.

HOW LACERO & BLOCKCHAINS INTEROPERATE

Ledgers are used to store information about an event, usually a contract. The full description of a contract or the content is usually too big to store inside a blockchain, Bitcoin transactions being an exception. Typically, content is stored outside the blockchain and an entry to access content is stored instead. This can be a simple identifier, an owner identifier or a mix of identifiers that may represent a hierarchy of ownership or access.

In the simple case, the object identifier and a hash of the content is stored in the blockchain as a minimum payload. Additional metadata can support a time stamp or other specific identifiers that collectively allow recovery of the content from a data source. Lacero creates and uses a metadata identifier and its hash.

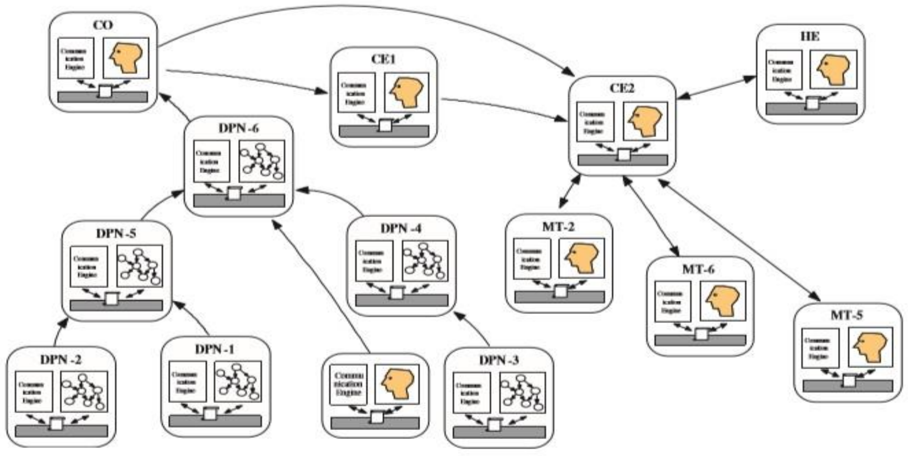
Through Lacero the metadata identifier is used to load from an external source the actual metadata that is used to query the storage nodes where the object fragments were stored to recover the content. Hashes play an important role through the Lacero component.

There are many variants on options available for metadata management but the important thing to remember is that metadata lives outside of *IOTA*^{F} which can find and reconstruct a sharded object from several storage environments.

THALES DPIF FRAMEWORK

IOTA^F is a standalone fabric for securely interconnecting devices and service using IPV6 addressing and its extensions. It permits the capture of events as transactions between one or more nodes and storing them in blockchain distributed ledgers. It also supports the storage of transaction content, or data objects in general, in a distributed, virtual, redundant, storage environment such that the stored data, in the form object fragments, is useless to anyone without the metadata needed to reconstruct the object.

Thales has developed a Dynamic Process Integration Framework (DPIF). It is a service oriented architecture supporting uniform encapsulation and combining heterogeneous processing capabilities required for fusion of large amounts of heterogeneous information. The processing capabilities can be provided by human experts or automated reasoning processes.

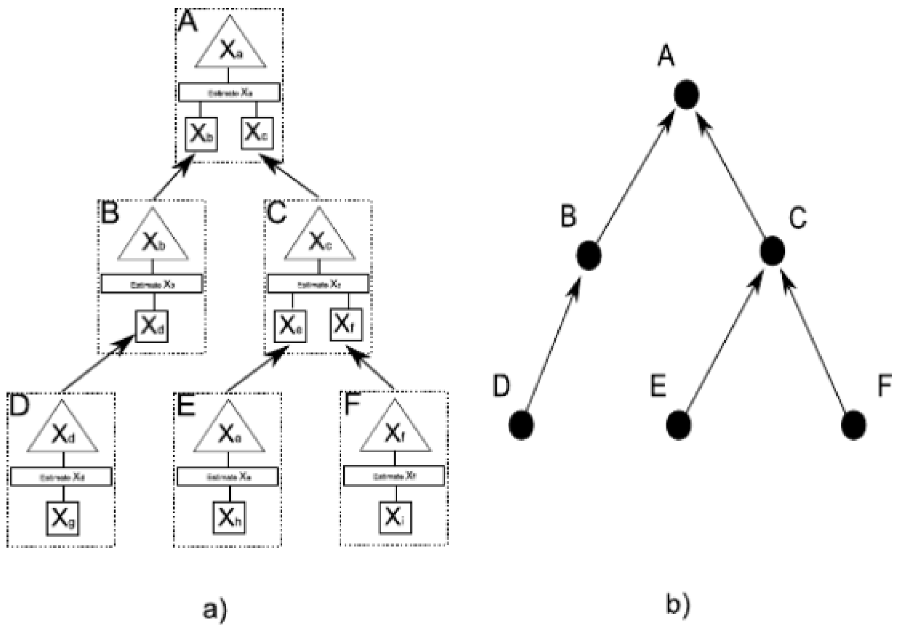


DPIF FRAMEWORK

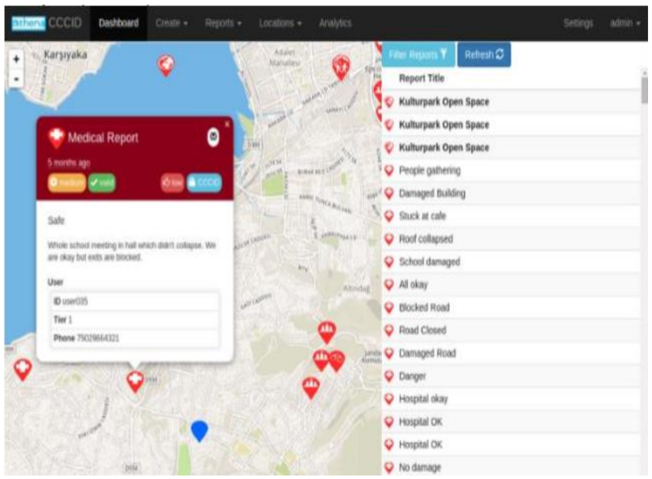
In the DPIF context, human expertise and automated processes are abstracted to functions with well-defined outputs and inputs where each function provides a service given certain inputs. DPIF provides function wrappers i.e., software agents with standardized interfaces. Agent interfaces are based on standardized service descriptions as well as uniform self-configuration, negotiation and logical routing protocols.

With the help of the DPIF encapsulation methods disparate services can negotiate to support systems at runtime. Moreover, DPIF agents support automatic formation of workflows in where functions can represent actors such as suppliers and consumers, outputs of functions as inputs to other functions, and so on. Contrary to typical approaches to runtime service composition, DPIF does not require any centralized ontology describing relations between services or configuration controls.

More important, in recognition of the challenges of data ownership (where it resides and where requestors exist) DPIF is designed to support data fusion without transfer of the source data to a central processing function.



The fundamental process is composed of individual workflows that connect from local domains of data and rules into a serialized workflow. In the diagram above (a) shows a self-organized system of agents. Each agent supplies information concerning a variable of interest in the domain. These outputs are based on other inferred or directly observed variables. Then (b) displays the relations in a directed GATEWAYgraph capturing the workflow between the agents.



Medical Report

5 months ago

Safe

Whole school meeting in hall which didn't collapse. We are okay but exits are blocked.

User

ID user035

Tier 1

Phone 75029664321

Report Title

- Kulturpark Open Space
- Kulturpark Open Space
- Kulturpark Open Space
- People gathering
- Damaged Building
- Stuck at cafe
- Roof collapsed
- School damaged
- All okay
- Blocked Road
- Road Closed
- Damaged Road
- Danger
- Hospital okay
- Hospital OK
- Hospital OK
- No damage

A relevant and more recent example of DPIF at work is in the ATHENA Mobile App designed for fire first responders.

The initial focus is to integrate *IOTA*^{F} fabric into DPIF to take advantage of its search and discover capabilities from the graph search function coupled with the of Intervault E2E security supporting both blockchain and Lacero distributed storage components.

The goal is to enable secure collaboration services between human agents as well as M2M and H2M devices and services. The effort will center on applications in **health management** based on the work of the creators in the medical sectors in the US and Japan as well as Thales efforts in the EU.

SUMMARY

IOTA fabric was created from well-established solutions for data collaboration, user and data, authentication, secure communications, and integration to third party and open source technologies.

These include graph search, end to end encryption, survivable storage systems, and recent advances such as permission-based blockchains.

The fabric design is focused on servicing the new Internet of Things/Everything (IOT/IOE) as the web rapidly adopts a P2P mesh network topology rather than its current server centric architecture.

The proliferation of IPV6 connectivity requires new approaches to finding, registering, connecting and auditing both devices and services.

We believe that IOTA uses the best of breed approach to open source and third party solutions, connecting them as a fabric, ensuring more secure Internet for business and government.