Bletchley – The Cryptlet Fabric & Evolution of blockchain Smart Contracts

Posted on February 9, 2017

Marley Gray
Principal Program Manager, Azure Blockchain Engineering

Anatomy of a Smart Contract

The concept of a Smart Contract has been around for awhile and is largely attributed to Nick Szabo's (https://en.wikipedia.org/wiki/Nick_Szabo) work in the late 1990s. However, it remained an abstract concept until the summer of 2015 with the Frontier release of Ethereum as its first implementation. The promise of Smart Contracts is sprawling and has gotten the attention of every industry as a revolutionary disrupter that can change the way business is done forever. That remains to be seen, but like most first implementations of significantly important technology, there are some early lessons learned and some introspection about how improvements can be made.

I have written a paper that describes at a high level how Smart Contracts are implemented today and how they can be refactored to significantly improve their performance, security, scalability, manageability, versioning, and reuse in the near future. This paper describes the thought process and

historical context for a new architectural approach that focuses on separation of concerns and implementing a 3 Layered/Tiered Smart Contract architecture.

To understand the context and exactly what a "3-Layered & Tiered" Smart Contract architecture means please give this paper a read, Anatomy of a Smart Contract (https://github.com/Azure/azure-blockchain-projects/blob/master/bletchley/AnatomyofASmartContract.md).

If you want the short and sweet answer it is this: Smart Contracts designed for semi-trusted enterprise consortium networks should be separated into 3 main layers:

- Data Layer – The definition of data schema and *only* the data logic for validation of inserts (appends) and optimization of reads. In platforms like Ethereum or Chain, languages like Solidity and Ivy can be used at this layer. This is similar to how relational databases use the SQL language and stored procedures.
- Business Layer – All business logic for Smart Contracts and surface level APIs for interacting with Smart Contracts from the Presentation layer (UI) or other external applications. Cryptlets written in any language targeting the runtimes supported by the Cryptlet Fabric. (.NET, .NET Core, JVM, native)
- Presentation Layer – User Interface platforms and other applications built on using the exposed APIs by Cryptlets.

These Layers can then be deployed, optimized, and scaled in their respective tiers: Presentation, Middle (Business), and Data tiers.

> **Note, this approach is not generally valid for trustless implementations of Smart Contracts, but targeted at enterprise consortium blockchains.*

The Cryptlet Fabric – Update

As we get closer to the first public preview of the Cryptlet Fabric, I though it was a good time to quickly highlight what blockchain enthusiasts, professionals and developers should be looking forward to. The Cryptlet Fabric is designed to be the middle tier in the new proposed 3-Layered & Tiered Smart Contract architecture, so it will deliver things you would expect from such an implementation. It manages scale, failover, caching, monitoring, management…a long list of features, but what is new:

Cryptographic primitives, secure execution enclaves and a runtime secure key (secrets) platform that allows for the creation, persistence and dynamic reanimation for performing cryptographically secure operations with these keys at scale.

This allows Cryptlets to create, store, and use key pairs in a secure execution environment to do all sorts of "blockchain-ey" things like digital signatures, zero knowledge proofs, ring signatures, threshold signatures, and even homomorphic encryption all within secure enclaves. Cryptlets will come in 2 basic types, which I went into some detail in the reading Cryptlets in Depth (https://github.com/Azure/azure-blockchain-projects/blob/master/bletchley/CryptletsDeepDive.md), utility and contract.

The Cryptlet Fabric provides a blockchain router layer, abstracting away different transaction messages and cryptographic signing from the code implemented in Cryptlets, as well as integration with existing systems. For example, a Cryptlet's output only contains the information to be sent to the blockchain, like market data or business logic results, and is packaged into the actual blockchain specific transaction by the Cryptlet Fabric below it by the blockchain router and blockchain platform specific providers. This encapsulation technique has been used for decades allowing technologies like TCP/IP to work on all sorts of networks (LAN, WAN, Internet and Mobile) and all kinds of applications. This is what allows for Cryptlets to be reused across different types of blockchains.

Utility Cryptlets (oracles)

Ok, I admit that being a Microsoft guy makes it hard for me to call anything an oracle without associating it with the company. The fact of the matter is that Utility Cryptlets can largely be thought of as a more scalable, standard, secure, and discoverable way to write blockchain oracles. The majority of requests that we get for Cryptlets are initially in this category, providing a secure attested data source for blockchain applications. You will able to write cryptlet oracles using the language of your choice targeting (initially) .NET, .NET Core, JVM and bare metal runtimes, so C#, Java, C++, F#, VB, etc... Your cryptlet oracle can publish any type of data you want, including:

- Market prices
- Package delivery notifications
- Workflow next steps
- Weather alerts
- Wounterparty updates like credit downgrade

and define your subscription parameters like:

- Time and date
- Event driven triggers from the blockchain or listening to external sources
- Conditional – if "this" and "that" are true, evaluation combinations if, else if, switches, etc.

- Combination – if the time is 4PM EST & the NYSE was open today then send me the LIBOR and the price of *x*

As a developer or provider of cryptlet oracles you can publish them as libraries into the Azure Marketplace for discovery and acquiring customers. We expect to see a robust catalog of Cryptlet libraries from our partners big and small.

However, there is more to Utility Cryptlets than just providing data. You can use them to expose a secure channel into your blockchain based applications for other applications and user interfaces. An ERP or CRM system can read data from the blockchain without having to know any of the details, and they can also securely fire data and events into the blockchain to ease integration of blockchains into existing enterprise infrastructure. Cryptlets can also provide services to blockchain applications and analytics platforms by triggering events into existing systems by watching blockchain transactions and issuing an index request or data pull.

I'm sure that customers and partners will find many uses for Utility Cryptlets, but the first and most obvious is for blockchain oracles.

Contract Cryptlets

These Cryptlets are key to fully realizing the refactoring of Smart Contracts into a 3-Layered/Tiered architecture. These Cryptlets differ from Utility Cryptlets in many ways. Utility Cryptlets can have many instances at once, each handling many different subscriptions at the same time, whereas Contract Cryptlets are single, paired, or ring instanced and bound to a specific Smart Contract. Contract Cryptlets contain the business logic, rules, and external API for interaction for executing a Smart Contract between counterparties. In a private, semi-trusted consortium blockchain network where there are strong identities or trust between counterparties, the Smart Contract logic does not need to exist "on-chain" or be executed by every node on the network. Separation of concerns between the data layer in the blockchain and the business and presentation above, isolates code between layers abstracting implementations for reuse and optimization.

Contract Cryptlets are themselves digitally signed and run within an enclave that attests that the code they contain ran unaltered without tampering in private. This allows for the code in the Cryptlet to be private between counterparties and even be encrypted to protect intellectual property. Counterparties can write and review their own Contract Cryptlets and all inspect and verify the code before compiling and packaging for implementation, or they can choose from certified Contract Cryptlets that a vendor or reputation stands behind.

Contract Cryptlets can be instantiated for each counterparty participating in a specific Smart Contract instance. These Cryptlet Pairs and Rings are optional, but allows for each Contract Cryptlet to hold secrets like private keys for signing or encryption belonging to that counterparty, avoiding co-mingling

secrets of counterparties in the same address space. This capability allows for more advanced transaction scenarios, like encrypting results, to be stored on the blockchain using threshold or ring encryption schemes.

Contract Cryptlets can use private keys and secrets it has permissions to that are fetched from the fabric's "secret" services. These secrets are controlled completely by the counterparty via Azure Key Vault, and remain inaccessible to other participants, including Microsoft.

This approach allows code defining a Smart Contract's business and integration logic to be run on scaled up resources and collocated near data for maximum performance. The data that they persist to the blockchain is still subject to the data logic "on-chain" and is reconciled on the network through its consensus process. The digital signatures from the Cryptlet, enclave and blockchain provider are used for validation and attestation, and can be stored along with each transaction on the blockchain as proofs.

Contract Cryptlets also use Utility Cryptlet services, however they can interact directly within the fabric via a subscription or direct API access.

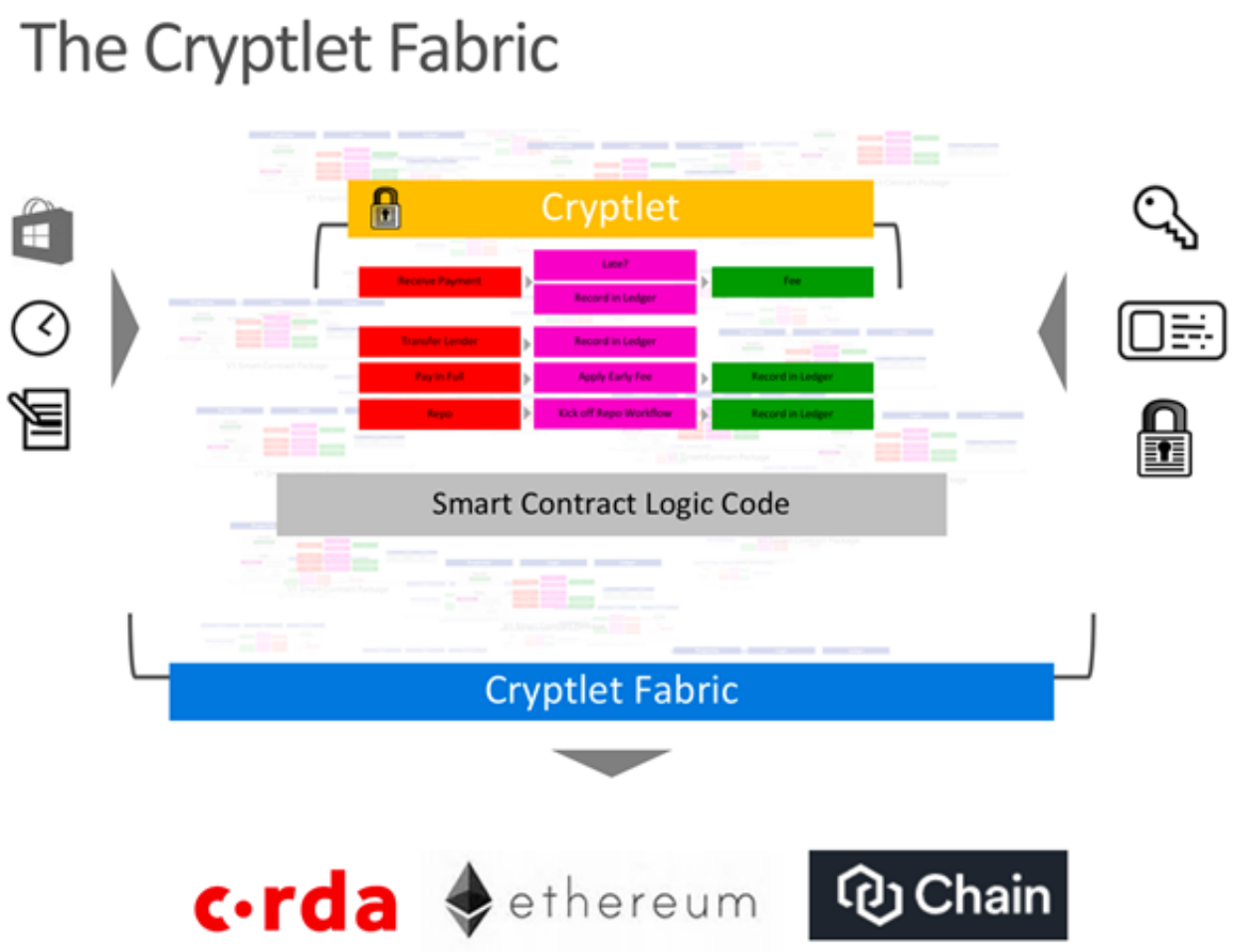
Using Cryptlets allows for a three layered architecture where:

- The Smart Contract on the blockchain defines schema and data logic, which is deployed to the blockchain network representing the Data Tier.
- Contract Cryptlets define business logic and a Surface Level API for User Interfaces and external applications deployed to the Cryptlet Fabric representing the Middle Tier.
- Utility Cryptlets provide reusable data sources and events also in the Middle Tier.
- Presentation tier communicates with the Surface Level API of the Cryptlet Fabric using standard UI technologies or integration platforms deployed to web servers, mobile devices, service bus, etc. for the Presentation Tier.

This architectural approach provides abstraction of the blockchain implementation from its clients, as well as tuning and scaling the blockchain and Cryptlet Fabric independently.

Cryptlet Fabric Diagram

Here is an updated diagram of the Cryptlet Fabric. We will cover its parts in more detail in subsequent posts and white papers, which will also be included in its releases.



(https://azurecomcdn.azureedge.net/mediahandler/acomblog/media/Default/blog/3cebfd6b-e136-49b8-aa5d-d83af1fd9404.png)

Blockchain (/en-us/blog/topics/blockchain/) blockchain (/en-us/blog/tag/blockchain/)

bletchley (/en-us/blog/tag/bletchley/) ethereum (/en-us/blog/tag/ethereum/) chain (/en-us/blog/tag/chain/)

corda (/en-us/blog/tag/corda/)

Topics

Announcements (/en-us/blog/topics/announcements/) (1388)

Big Data (/en-us/blog/topics/big-data/) (350)

Blockchain (/en-us/blog/topics/blockchain/) (27)

Business Intelligence (/en-us/blog/topics/business-intelligence/) (18)

Cloud Strategy (/en-us/blog/topics/cloud-strategy/) (411)

Data Warehouse (/en-us/blog/topics/data-warehouse/) (45)

Database (/en-us/blog/topics/database/) (280)

Developer (/en-us/blog/topics/developer/) (706)

Events (/en-us/blog/topics/events/) (56)

Government (/en-us/blog/topics/government/) (14)

Identity & Access Management (/en-us/blog/topics/identity-access-management/) (66)

Internet of Things (/en-us/blog/topics/internet-of-things/) (53)

IT Pro/DevOps (/en-us/blog/topics/it-pro-devops/) (407)

Management (/en-us/blog/topics/management/) (143)

Media Services & CDN (/en-us/blog/topics/media-services/) (144)

Mobile (/en-us/blog/topics/mobile/) (141)

Networking (/en-us/blog/topics/networking/) (102)

Security (/en-us/blog/topics/security/) (132)

Storage, Backup & Recovery (/en-us/blog/topics/storage-backup-and-recovery/) (409)

Supportability (/en-us/blog/topics/supportability/) (17)

Updates (/en-us/blog/topics/updates/) (48)

Virtual Machines (/en-us/blog/topics/virtual-machines/) (372)

Web (/en-us/blog/topics/web/) (296)

Articles by date

January 2017 (/en-us/blog/2017/01/)

December 2016 (/en-us/blog/2016/12/)

November 2016 (/en-us/blog/2016/11/)

October 2016 (/en-us/blog/2016/10/)

September 2016 (/en-us/blog/2016/09/)

August 2016 (/en-us/blog/2016/08/)

Full archive (/en-us/blog/archives/)