



Software Identification and Entitlement Tracking Using Blockchain Technology

Citation

Swanson, Jared. 2018. Software Identification and Entitlement Tracking Using Blockchain Technology. Master's thesis, Harvard Extension School.

Permanent link

<https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37364546>

Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA>

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. [Submit a story](#).

[Accessibility](#)

Software Identification and Entitlement Tracking Using Blockchain Technology

Jared Swanson

A Thesis in the Field of Information Technology
for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2018

Copyright 2018 Jared Swanson

Abstract

This research assessed the applicability of blockchain technology to a registry for software asset management data aligned with the requirements of the International Organization for Standardization (ISO). ISO has developed a series of standards that specify data formats and practices for digital tags to help track software assets, but to date, adoption of the standards have been slow, and tools to improve tag deployments and utilization have not yet become widespread. To assess the potential for blockchain technology to catalyze the emergence of capabilities based on software tagging, this investigation involved a design and proof of concept implementation of a software tagging registry based on blockchain technology.

The preliminary research suggests that blockchain technologies could be very effective at enabling much higher degrees of security and automation for software tracking. Blockchains have proven highly effective for environments with high security demands, with low trust, and that require a full enumeration of all past changes. These traits closely match the characteristics of a software tag registry, namely a need for a transparent, publicly accessible, high-availability, platform to share tags in an environment with diverse methods for determining trust.

The proof of concept was implemented using a Hyperledger Fabric blockchain platform, which was mostly successful. However, the exercise revealed several key weaknesses of the Hyperledger Fabric blockchain implementation for that application. These weaknesses were specific to the Hyperledger Fabric implementation of blockchain,

and do not apply to blockchain technologies in general. The research and proof of concept indicated that using blockchain to implement a software tagging registry is a viable approach, but that development of a full registry would likely require a different blockchain implementation than the one used in this proof of concept.

Contents

Abstract.....	iv
Contents.....	vi
Figures.....	viii
Chapter 1: Introduction.....	1
Chapter 2: Background.....	6
2.1 SWID Tags.....	8
2.2 Entitlement Schema.....	12
2.3 Blockchain.....	15
2.4 SWID, Ent, and Blockchain Integration.....	17
Chapter 3: Software Registry Concept.....	20
3.1 Trust Relationships.....	22
3.2 Financial Feasibility:.....	24
Chapter 4: Competing Capabilities.....	26
Chapter 5: SWID and Ent Registry Design.....	28
5.1 Key Architectural Components.....	28
5.1.1 Cloud Server (IaaS).....	28
5.1.2 Docker Platform.....	29
5.1.3 Blockchain Solution.....	29
5.1.4 SWID/Ent Channel Smart Contract Functions.....	30
5.1.5 SWID/Ent Gateway Interface.....	30
5.1.6 Web Front End.....	31
5.2 Selection of Hyperledger Fabric:.....	31
5.3 Hyperledger Fabric Architecture.....	32
5.3.1 Key Concepts.....	33
5.3.2 Types of Nodes.....	34
5.3.3 Transaction Time Sequence Diagram.....	34

5.4 Roles and Use Cases.....	36
5.4.1 Supported Roles and Use Cases.....	36
5.4.2 Unsupported 19770-3 Use Cases.....	39
Chapter 6: SWID and Ent Registry Development.....	42
6.1 Implementation Process.....	42
6.1.1 Environment Setup.....	43
6.1.2 Hyperledger Fabric Network.....	43
6.1.1 Smart Contract for the SWID Blockchain.....	44
6.1.2 Smart Contract for the Ent Blockchain.....	47
6.1.3 ExpressJS Server.....	48
6.1.4 Graphical Web-Based Interface.....	48
6.1.5 Deployment to Cloud.....	50
6.2 Review of Hyperledger Fabric Implementation.....	50
6.2.1 Hyperledger Strengths.....	51
6.2.2 Hyperledger Weaknesses.....	53
Chapter 7: System Walkthrough.....	56
Chapter 8: Future Capabilities.....	64
8.1 Peer-to-Peer Information Sharing.....	64
8.2 Certificate Authority for Trusted Software.....	65
8.3 Reputation Scoring.....	66
8.4 Software Whitelisting on Endpoints.....	67
8.5 Software Discovery.....	68
8.6 Decomposition of Application Dependencies for Digital Rights Assurance...72	
Chapter 9: Conclusion.....	73
References.....	75
Glossary.....	77

Figures

Figure 1. SWID Tag UML Diagram.....	11
Figure 2. Ent UML Diagram.....	14
Figure 3. Minimal SWID and Ent Data Required by ISO 19770 Standard.....	21
Figure 4. Key Components of SWID and Ent Registries.....	28
Figure 5. Sequence of Successful Hyperledger Fabric Transaction.....	34
Figure 6. Implemented Use Cases.....	36
Figure 7. Supported and Unsupported ISO Standard 19770-3 Use Cases.....	40
Figure 8. Pseudo-UML Diagram of Validator Design.....	46
Figure 9. Graphical User Interface Page Functions.....	49
Figure 10. Graphical User Interface Home Page.....	57
Figure 11. Graphical User Interface SWID Submission.....	57
Figure 12. SWID Tag Submission Sequence Diagram.....	58
Figure 13. Successful Tag Submission.....	59
Figure 14. Informative Error.....	59
Figure 15. SWID Tag Query Page.....	60
Figure 16. Initial Ent Tag.....	61
Figure 17. Ent Submission Success 1.....	61
Figure 18. Ent Decrementing Entitlements 1.....	61
Figure 19. Ent Submission Success 2.....	62
Figure 20. Ent Decrementing Entitlements 2.....	62
Figure 21. Ent Submission Success 3.....	62
Figure 22. Ent Decrementing Entitlements Below Zero.....	62
Figure 23. Ent Submission Informative Error.....	63
Figure 24. Ent Query Page.....	63

Chapter 1: Introduction

Software asset tagging has great potential to improve software asset management, cybersecurity, and software license management. The ability to tag all trusted software would facilitate the development of a range of tools that could greatly improve current software asset management and cybersecurity methods.

The International Organization for Standardization (ISO) developed standards for software tagging that targets software asset management as its primary use case. It specifies where tags can be stored, what minimal set of information is required in each tag, how the data should be formatted, and what standard operations are permissible for tags. This greatly facilitates software asset management practices by permitting scanning tools to perform scans of an environment to discover all software very quickly and potentially with perfect results. Tags can also be used to track software licenses and to associate each instance of software with the appropriate license to ensure that all software in the environment is operating with a valid license.

In terms of cybersecurity, software tagging could provide a much simpler and more reliable approach if it is coupled with trust-based policies or trust-related data to serve as a basis for software whitelisting. Blacklisting is still a prevailing security method used in the cybersecurity industry, but they are not able to address modern attack vectors such as polymorphic malware or zero-day attacks. Polymorphic malware is a malicious form of software that can continually change its own code to evade detection, but preserves its ability to perform its intended function. A zero-day attack is an exploit of a

software vulnerability that is not known to its potential victims, thereby making it difficult to prevent.

The inadequacy of blacklisting technologies has led many current state-of-the-art technologies to adopt behavior-based approaches. Behavior-based tools identify behaviors that are considered normal within an environment, and then react to departures from normal system behavior. Behavioral techniques can be great additions to a defense-in-depth strategy, but unfortunately, as behavioral technologies become more accurate and sensitive, attackers are able to devise attacks with increased stealth to blend in with normal operations. Further refinement of behavioral tools or layering behavior-based security with additional security measures increases complexity, which can itself introduce new costs and new vulnerabilities. Simple software whitelisting approaches using information stored in a list or database of trusted tags could disrupt the current trend with a much simpler implementation that could even ensure that every bit of each program on a system is trusted.

Properly tagged software can also be used to easily correlate matching licenses to ensure complete compliance with licensing requirements across an enterprise. Licenses could then be updated or removed as necessary. Most tools currently performing this task require manual input and tracking, and many rely on warnings for product deactivations, or reminder emails from vendors to renew software licenses. An enterprise with comprehensive tagging could track software deployments internally, better optimize its license allocations, and achieve much greater degrees of automation in the license management process.

While useful in theory, software asset tagging has been slow to mature and fulfill the role its creators envisioned for it in cyberspace. For one, the cases described above assume full tagging of cyber assets. It is likely that no single enterprise has tags for all of its software, and it would be prohibitively expensive for organizations to create the tens of thousands of tags necessary to enable them to use tag-based tools to account for the software components they use. Since customers are not using tools that require software asset tags, software vendors see little need to begin tagging their products. This creates a situation where neither vendors nor consumers have the incentive to initiate the creation of a body of software tags.

Blockchain technologies may provide a means to overcome the economic gridlock preventing software tags from achieving more widespread use. The concept of a blockchain is still not rigorously defined, but it is often associated with a concept of a distributed ledger, which is a set data shared between many stakeholders, and a defined set of actions stakeholders can take to modify the data in the ledger. Blockchains are a relatively new technology that shows promise to allow transactions to be conducted in environments where all participants are distrustful of one another.

Technology based on a blockchain could be used to crowdsource a registry of software tags to distribute the costs of tagging software. Tags placed on a publicly accessible blockchain could allow users to trust records with selected digital signatures, even if they don't trust any other users of the blockchain, or even the providers of blockchain services. This means that organizations would no longer need to tag their own software if a trusted tag is already available in the blockchain. They could use records from any trusted tag creators, and issue new tags for a much smaller subset of software.

Once the blockchain matures and the data becomes more comprehensive, most organizations may be able to completely automate the process of software tagging.

The validation of content from websites is an example of a mature capability where cryptographic signatures are used for certificate authorities to vouch for the credibility of websites. Certificate authorities sign and store trusted public keys so users can validate the credibility and authenticity of the website they are visiting. This service is transparent to most users, but has evolved into a multi-billion dollar industry. There is currently no comparable infrastructure that will automatically certify programs or applications stored on a user's machine. However, signed tags can be used to perform software assurance, and a blockchain with signed software tags could provide the backbone for this type of capability.

Blockchains have also been proposed and implemented as tools to provide attestation for copyright or ownership of any software at a certain point in time. By necessity, blockchains must determine a specific order for all of its data. This requirement can be used to provide high-integrity timestamps for any records added to the blockchain, which can then be validated by anyone who can access a copy of the blockchain. Therefore, even the most basic blockchain implementation for software asset tags could extend to serve this, and many other functions. It is important to note that the ISO software tagging standards are intended to apply to a much broader set of software components than executable files and applications. They are intended to enable tagging for *all digital assets*. This may include authors tagging their written works, musicians tagging their songs, researchers tagging their databases, and a vast number of other use cases.

The research conducted in this thesis was the creation of a blockchain platform to serve as a software tag registry. The intent of the registry, if deployed publicly, would be to overcome the low adoption of software tagging by using a crowdsourcing strategy to create a central, publicly accessible database to permit tag reuse throughout the cyber ecosystem.

Chapter 2: Background

Many organizations struggle to track software product deployments and locate software assets across their enterprises. After decades of industry evolution many companies are still surprised by the results of software audits, the size of bills paid for unused software, or breaches due to outdated and forgotten software running on their systems. These challenges are a result of ineffective software asset management processes and capabilities.

Software asset management has not drawn the same level of interest and innovation as other technologies like big data, cyber threat analytics, machine learning, and other higher-profile fields. Yet the need to track software assets and licenses continues to be crucial for many organizations. In recognition of this need, the International Organization for Standardization (ISO) has created the 19770 family of standards to define software asset management (SAM) processes and formats. Over the past two years, these new standards have presented new formats and practices for more effective management of software assets and tracking, but adoption of these standards into industry tools and processes has been slow. Yet, the slow maturation of SAM capabilities is not due to a lack of need.

According to a 2016 Gartner report, many organizations can reduce spending on software up to 30 percent through application configuration optimization, recycling software licenses and SAM tools. That year Gartner estimated that organizations would spend \$332 billion on software (Gartner, 2016).

While organizations continue to demand better software asset management, a review of existing literature and existing capabilities show that the processes remain unreliable, and labor intensive. Efforts in standardization and cyber information sharing could provide a solid foundation to increase the quality and reduce the cost of software asset management efforts. The Department of Defense provides one example of movement in this direction. The DoD Information Technology Standards Registry (DISR) is the single DoD registry for approved information technology standards, and standards from the DISR Baseline are mandated for use in the DoD acquisitions process. (DoD Information Technology Standards Registry, 2017) The DISR Standards and guidance lists the ISO/IEC Standard 19770-2 as “Mandated.” According to the DoD definitions, this means that it “must be used in lieu of a competing or similar standard.” and that “The standard is required for the management, development, and acquisition of new or improved DoD systems that produce, use, or exchange information.” ISO/IEC Standard 19770-2 defines an internationally-recognized format and set of data elements for use in digital Software Identification tags (SWID tags). The DoD requirement to use tags compliant with the ISO 19770-2 Standard is one sign that large organizations recognize the value of these tags, and that they are likely to become more prevalent in the marketplace in the coming years.

These developments may provide a strong foundation to build utilities that could support more capable and efficient software asset management programs. SWID tags, combined with their sister standard for a software entitlement schema, and new advancements in blockchain technology could work together to transform the software asset management industry.

2.1 SWID Tags

ISO Standard 19770-2 defines a common format for expressing information about software products and a set of operations to record and track relevant changes to software over time. These are known as Software Identification tags, more commonly called SWID tags. The first version of ISO Standard 19770-2 was published in 2009. An updated version was published in 2015 to improve upon the original standard.

To use this software tagging standard, organizations need to issue, replace, and remove software identification (SWID) tags as software is deployed, updated, and removed. While there are some tools that use SWID tags for software discovery, there are no commercially available tools to largely automate tag deployment and analysis.

The infrastructure for software tagging is not sufficiently mature for most software consumers to justify investments in SWID-based technologies. The ISO 19770-2 Standard recognizes that software publishers are best positioned to tag their own products; however, software publishers lack a sufficient incentive to drive widespread availability of SWID tags for software products and components. This means that any investment in SWID-based discovery technologies on the part of software consumers would only grant visibility to a small number of software packages, which would not address organizations' needs for security or SAM solutions. As SWID tags become more prevalent, investments in these technologies become better justified.

There is strong evidence that the use SWID tags is growing more prevalent. Corporations like Microsoft, Symantec, Adobe, Hewlett Packard, and IBM have all signed on to participate in efforts related to SWID tagging. The U.S. Federal Government

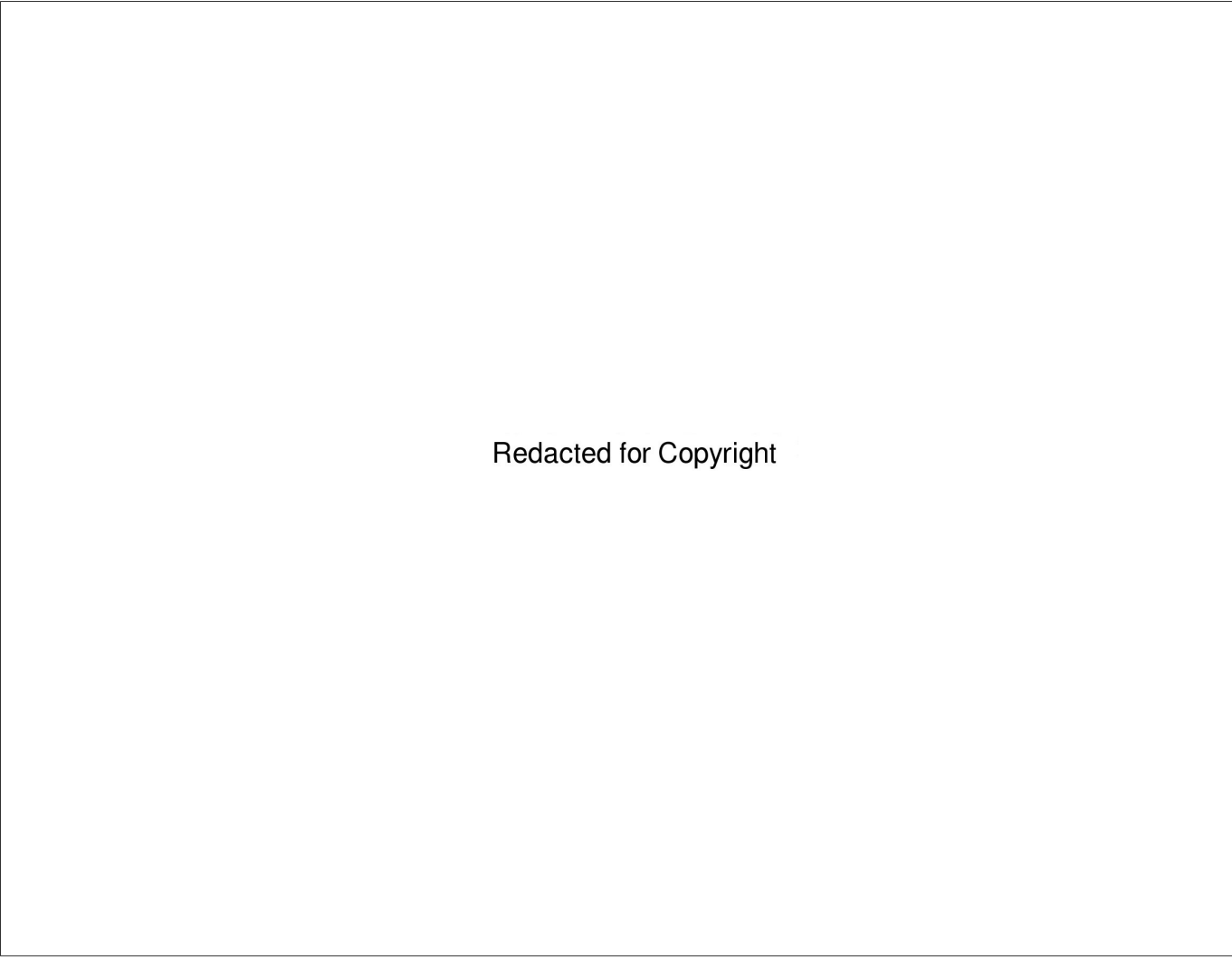
is also taking steps to drive adoption in the marketplace. In April of 2016, the National Institute of Standards and Technology (NIST) published guidance on SWID tag usage for the Federal Government in its Interagency Report 8060 titled “Guidelines for the Creation of Interoperable Software Identification (SWID) Tags” which adds additional government-centric requirements on top of the existing ISO Standard 19770-2 requirements. The U.S. Army, U.S. Navy, U.S. General Services Administration, U.S. Department of Homeland Security, National Institute of Standards and Technology, and Department of Defense have all participated in efforts to drive SWID tag adoption. This level of backing by large organizations in the software marketplace suggests that SWID tags will grow more prevalent over time.

If ISO 19770-2-compliant SWID tags become more prevalent, investment in SWID tag-based SAM practices could prove highly beneficial to many organizations.

They could more easily:

- 1) Purchase diverse standards-compliant tools which interoperate with tools in their existing infrastructures
- 2) Gain better and more granular insight into the set of software assets they use than is possible with current technologies
- 3) Precisely locate outdated and insecure software
- 4) Avoid vendor lock-in, (an anti-competitive situation that occurs when an organization tailors its infrastructure too closely to specific vendors’ products, and can only switch to incorporate products from other vendors at great cost)
- 5) Centralize, track, and optimize software deployments. For example, unused devices, or unneeded software deployments can help an organization identify its unused and unneeded assets and reduce its consumption of software products
- 6) Remain flexible amid changing technologies, infrastructures, and legal requirements.
- 7) Take advantage of industry best practices and share lessons learned that emerge surrounding the standard
- 8) Use external auditing to ensure proper system implementation
- 9) Meet the information needs of required audits with greater precision, and less time and cost

This list is not exhaustive, but is meant to illustrate tangible value from SWID tag-based SAM. A UML diagram detailing the structure of SWID tags as defined by ISO is shown on the next page.



Redacted for Copyright

Figure 1. SWID Tag UML Diagram. This material is reproduced from ISO 19770-2:2015 with permission of the American National Standards Institute (ANSI) on behalf of the International Organization for Standardization (ISO). No part of this material may be copied or reproduced in any form, electronic retrieval system or otherwise or made available on the Internet, a public network, by satellite or otherwise without the prior written consent of the ANSI. Copies of this standard may be purchased from the ANSI, 25 West 43rd Street, New York, NY 10036, (212) 642-4900, <http://webstore.ansi.org>.

2.2 Entitlement Schema

ISO Standard 19770-3 defines an entitlement schema intended to help track software entitlements aligning to ISO Standard 19770-1, which covers more general software asset management processes. These two software asset management standards are intended to maximally align to ISO Standard 19770-2, creating an integrated, cohesive structure for identification and management of software assets. It also seeks to incorporate existing entitlement information so existing entitlements can be easily ported into the software asset management process.

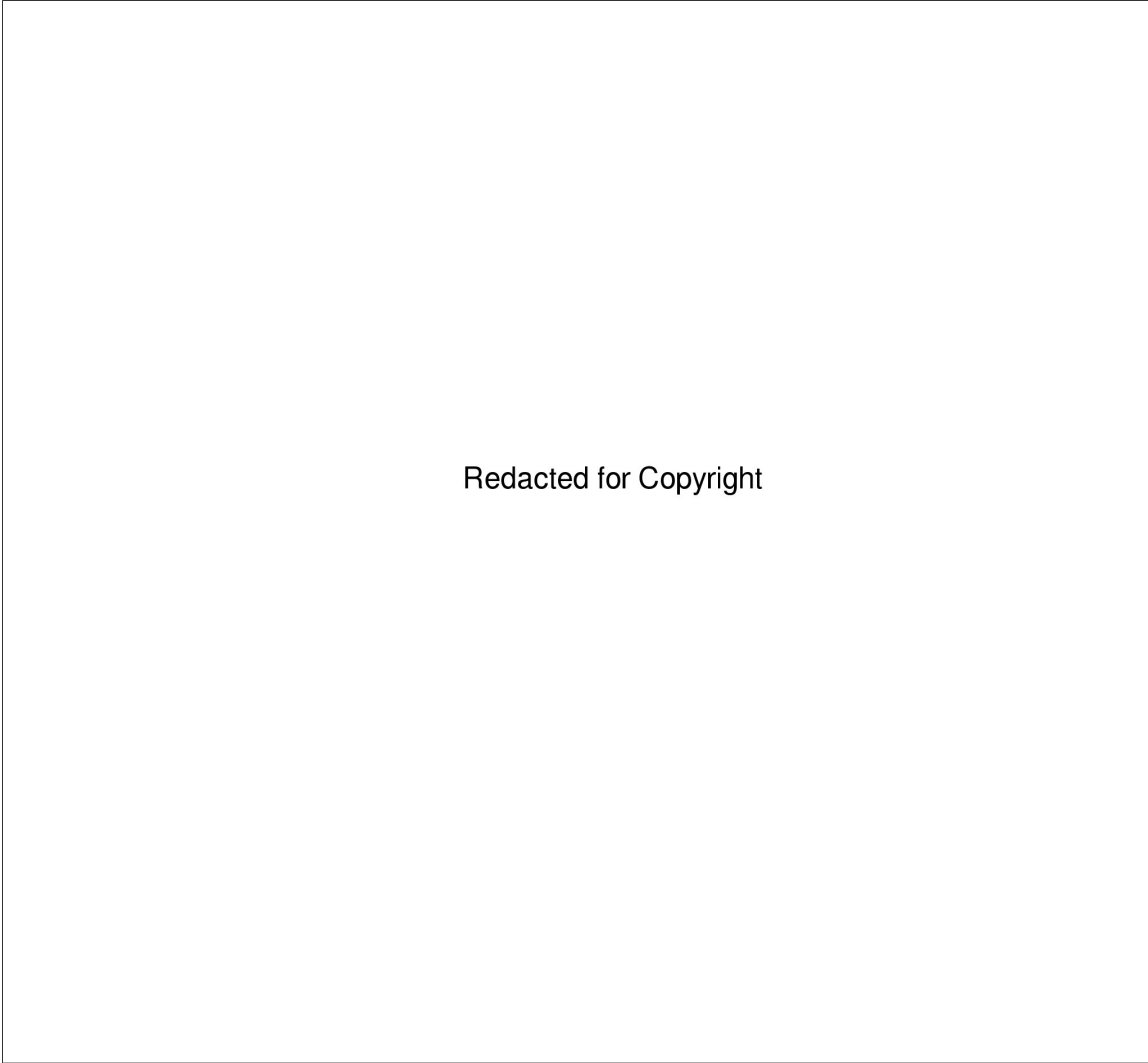
A software entitlement is a right to use a specific software product or service. Entitlement information stored in a data tag defined by the ISO Standard is referred to as an “Ent.” ISO Standard 19770-3 also notes a large number of benefits from using Ents. The most important regarding this research topic are:

- “immediate software consumer recognition of details of the usage rights derived from their software entitlement;
- “ability to specify details to customers that allow software assets to be measured and reported for license compliance purposes;
- “increased awareness of software license compliance issues on the part of end-customers;
- “improved software consumer relationships through quicker and more effective license compliance audits;
- “receipt of consistent and uniform data from software licensors, resellers, and SAM tools providers.
- “more consistent and structured entitlement information, supporting the use of automated techniques to determine the need for remediation of software licensing;
- “improved ability to avoid software license under-procurement or over-procurement with subsequent cost optimization;
- “standardized usage across multiple platforms, rendering heterogeneous computing environments more manageable” (ISO, 2016).

By linking entitlements to the SWID tags, contractual and financial information can, through automated tools, be combined to enable a complete mapping of software entitlements that an organization has purchased to their deployment across networks. Expanding from a system focused specifically on software identification based on SWID tags, which can be helpful for security functions, and adding entitlement information can provide a much greater depth of functionality. The main focus for the future evolution of ISO Standard 19770-3 seeks to determine the level of software usage so that software investments can be better optimized.

SWID tags and Ents share a very useful feature, often referred to as “decomposability.” This means that tagging can be conducted at many different levels. Tagging at lower, more detailed levels can achieve highly granular visibility into the software terrain of an organization. On the other hand, many software packages, such as a complete operating system, can be bundled into a single tag. This may be helpful for asset tagging at the device level. Tags can also cover bundles of software, such as suites of products, and they can tag each individual product, and even tag down to the level of each component in a software product. This capability provides a powerful and scalable model for tracking software assets, even down to the level of single files.

The UML domain model for SWID tags as defined by ISO is shown on the following page.



Redacted for Copyright

Figure 2. Ent UML Diagram. This material is reproduced from ISO 19770-3:2016 with permission of the American National Standards Institute (ANSI) on behalf of the International Organization for Standardization (ISO). No part of this material may be copied or reproduced in any form, electronic retrieval system or otherwise or made available on the Internet, a public network, by satellite or otherwise without the prior written consent of the ANSI. Copies of this standard may be purchased from the ANSI, 25 West 43rd Street, New York, NY 10036, (212) 642-4900, <http://webstore.ansi.org>.

2.3 Blockchain

Blockchain technologies first drew attention with the release of Bitcoin on January 9, 2009 by an individual using the name Satoshi Nakamoto, though the individual's actual name is unknown (Swan, February 2015). Bitcoin is the first known functioning blockchain application based on peer-to-peer file sharing using BitTorrent and common public key cryptographic techniques. Within a decade the market cap of the Bitcoin cryptocurrency has grown, peaking at over \$300 billion in December of 2017 (Blockchain.info, January 2018). The ability of a digital currency to succeed in low-trust environments, and remain secure and trusted despite antagonism from many actors, some with nation-state capabilities, has drawn broad interest from across software communities.

The most notable architectural characteristics of the Bitcoin architecture is: first, its decentralized method to achieve consensus, and second, the blockchain itself in the form of an immutable ledger that ensures the integrity of all Bitcoin transactions. Many engineers have used these concepts to create many new new digital currencies, though none have proven nearly as successful as Bitcoin.

A second characteristic of the Bitcoin blockchain has led to the emergence of a new concept known as a smart contract. Nakamoto developed Bitcoin with support for a specific set of contracts, including escrow transactions, bonded contracts, third-party arbitration, multiparty signatures, and others. The idea that sets of contracts could be written explicitly in code demonstrates great promise for automating transactions with greater speed, efficiency, and explicitness. This idea was expanded with the roll-out of the

Ethereum blockchain created by Vitalik Buterin in July 2015. Instead of building support for many types of contracts, as Nakamoto did for Bitcoin, Buterin created a set of Turing complete operations that could run on a blockchain. This concept would allow any contract that can be run on modern computers to be written and executed on the Ethereum blockchain.

Since the release of the initial Bitcoin and Ethereum blockchains, many new implementations and designs of blockchain technology have emerged, each design targeting a different set of challenges. Several of these implementations have become a part of the Hyperledger Project under the Linux Foundation. These include Hyperledger Burrow, contributed by Monax with co-sponsorship from Intel; Hyperledger Fabric, contributed by IBM; Hyperledger Indy, contributed by the Sovrin Foundation; Hyperledger Iroha, contributed by Soramitsu, Hitachi, NTT Data, and Colu; and Hyperledger Sawtooth, contributed by Intel.

In general, blockchains function well in low-trust environments. In many implementations any user can validate the entire history of transactions if they choose to. This openness helps the system maintain a high level of security compared to other database technologies, and provides current and potential users with proof of the trustworthiness of the database. Furthermore, it can be decentralized, which means that even if certain nodes hosting the database are taken off line, the network can continue to function. If necessary, every user could hold a record of all of the blockchain's data, and still be sure through cryptographic signatures that they retain records that are fully consistent with everyone else's blockchain.

While many blockchain implementations seek to address a range of problem sets, blockchains have drawn special attention for its potential to improve asset tracking and supply chain research. The research in this thesis fits that mold, and seeks to lay the foundations for greater interoperability between any tools or utilities that comply with existing international standards and that incorporate industry best practices for software asset management.

2.4 SWID, Ent, and Blockchain Integration

The structure of ISO Standard 19770-2 and 19770-3 lend themselves very well to an immutable data storage structure. Both SWID tags and Ent's are intended (though not required) to reference one another, making it possible to trace the history of any updates or changes to records to be through all related tags that have been previously deployed. Each set of related tags starts from a Primary Tag. Thereafter, additional tags describe changes or updates to the information of a previous tag. In the case of SWID tags, this includes installation, patching, new editions, software bundling. In the case of Ent's, the establishment of ownership of software is followed by many actions, including licensing, license renewal, and software asset transfers, each of which can be traced back to the first record specifying entitlement and licensing information for a software product. This decision to store software data as a Primary Tag and store a record of each change to that record makes software tagging highly amenable to immutable data storage design patterns.

Blockchains excel at assuring the integrity of its records using a design pattern that retains a record of any initial data and retains a sequential record of all changes. The

design of its integrity protection doubles as a method to allow users to trace the history of all transactions back to its earliest records, closely mirroring the way records in the ISO SAM standards are intended to be traced back to the earliest related record.

As a registry performing a security function, availability of the system is also important. Blockchains are often designed to be decentralized, meaning that they have no single points of failure. Many parties participate in maintaining and growing the data, and it would be difficult to compromise a sufficient number of nodes at a similar time to corrupt the blockchain. Generally, a blockchain with a larger number of participants will be more secure than blockchains with fewer participants. This method of using blockchain can make blockchains more secure than traditional databases which can be rendered unavailable by the compromise of a single node.

If blockchain served as the backbone technology for a registry for SWID tags and Ents, users would have an incentive to download their own copies of the ledger. In some contexts, they will want to hold a copy within their environments where their applications could access data more quickly, and where they could ensure they can still access the data in the case of an external network failure. Other users may want to intentionally validate the integrity of their tags. The incentive for many users to hold separate copies dovetails closely with the property of many blockchains where more users holding a copy of the blockchain increases the security of the data held within the blockchain.

In addition, blockchains can be designed to be pseudo-anonymous. In this case, identities are tied to cryptographic keys, rather than to individuals. Users can then use a private key of their choosing which may not be traceable back to the individual or organization that controls the key. For SWID tags and Ents, this could be useful to ensure

that people or organizations that want to provide information are permitted to do so without having that information traced back to them. For example, if a tag consumer wants to purchase new software, it would not want its competitors to know about its transactions and would want to remain anonymous, and so could choose to use a single-use key. Other blockchains are designed for users to have known identities that are linked to private keys. An example of this case would be a software creator that wants to sign tags for all of its products and patches so that its customers can validate the integrity of all of the software they buy and/or update. Both of these models can be used on the same blockchain instantiation.

Blockchain technologies are a concrete technology that naturally implements many of the needed capabilities to implement the ISO 19970 Standard. While many additional technologies are required to devise working and valuable SWID tag and Ent registries, a blockchain implementation is a logical tool to consider as a basic building block.

Chapter 3: Software Registry Concept

The centralization of SWID tags into a decentralized registry can allow users to leverage records that other individuals or organizations have previously created to pre-populate much of their software asset information that they may otherwise need to enter and track manually. Organizations providing tags may be software creators, commercial tag providers, or tag consumers. For software consumers, software tagging is usually prohibitively expensive, but by sharing information in a registry, their workload can be distributed between multiple parties. The ISO Standards for SAM provide an excellent opportunity to design a solution that is both universal and vendor agnostic. Any user is empowered to fill gaps by tagging any software, which continues to provide additional data that could make the registry more comprehensive, allowing any users to achieve higher levels of automation for their software asset tracking. This type of information sharing platform could reduce the costs of software asset tagging and provide a free and valuable resource for organizations to maintain a deep understanding of how they use their software resources. Registries of software identification and entitlement tags could then function as a foundational capability on top of which a larger and more complex set of systems, utilities, and business logic can be built.

The minimum set of data required for a SWID tag is fairly small:

- Software Identity
 - Name
 - Tag ID
 - Flag to Designate Tags for Patches

- Flag to Designate Supplemental Tags
- Version of the Tag
- Version of the Software
- Scheme off the Version (e.g., ‘multipartnumeric’)
- Entity
 - Role of Tag Creator
 - Regid of Tag Creator
 - Name of Tag Creator

For Ents the minimum set of required fields is:

- Ent
 - Ent Identifier
 - Ent Creation Date
 - Ent Type
 - Entity
 - Role of the Ent Creator
 - Regid of the Ent Creator
 - Name of the Ent Creator

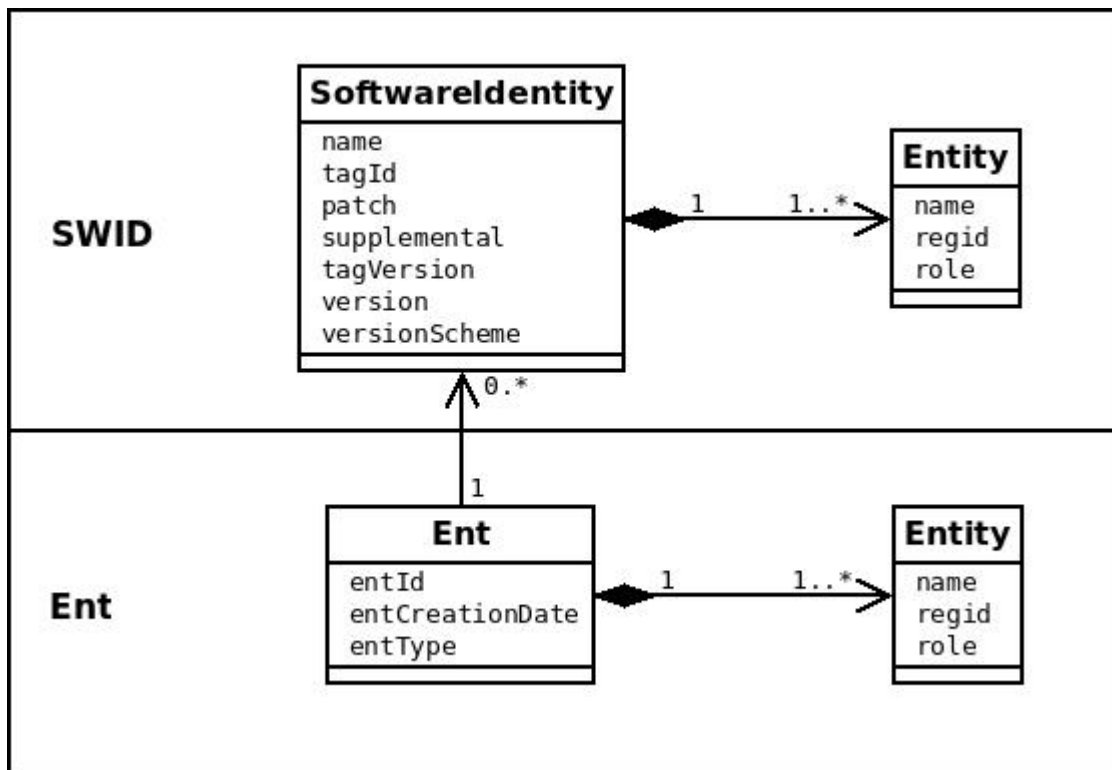


Figure 3. Minimal SWID and Ent Data Required by ISO 19770 Standard

SWID tags are infinitely extensible and can incorporate any relevant data. With a little additional information, stored either in the tags themselves or from an external source, software asset and license management tools could support a large number of capabilities that are currently in high demand. For the initial proof of concept, it would be better to ensure that the system remains extensible, as there are likely many more potential uses and systems that can be built on top of the registry than are envisioned here. That said, some examples may illustrate the power and diversity of the potential for new capabilities.

- Peer-to-peer information sharing
- Use of certificate authorities to validate trusted software
- Reputation scoring
- Software whitelisting
- Identification of unknown programs
- Decomposition of application dependencies for digital rights assurance

Any software creator could freely access and leverage the SWID and Ent resources to develop a plethora of new technologies. Adherence to international SAM standards could help the market to coalesce around a common, open data format to support the maturation of the industry.

3.1 Trust Relationships

Information sharing for SWID tag information can fit well into emerging resources and institutions operating in cyberspace. Notably, Executive Order 13691 called for the creation of Information Sharing and Analysis Organizations to help cross-company and cross-sector collaboration and information sharing for cybersecurity purposes (EO 13691, 2015). Since the evidence of a software program's presence will

likely be limited to a finite set of programs, ISAOs or similar commercial or 501(c)(3) organizations can publish information to a centralized blockchain for consumption across a broad number of organizations. The availability of a centralized data source may prove a defining factor that could make SWID tag-based SAM feasible in an environment where a relatively low proportion of software is tagged by the publisher or the distributor.

In the case of profit-seeking organizations, there will likely be a demand to limit access to blockchain information to paying subscribers only. These cases can all be covered in a single blockchain instantiation.

Information on a blockchain can be matched to a cryptographic public key, which can allow client applications to distinguish between trusted and untrusted tags. This enables a high degree of automation and a large range of configuration options to discriminate between trusted and untrusted data.

One trust model entails an organization choosing a list of tag issuers to trust. These may be software creators, or other third parties that issue software tags which have created tags for software products. This relationship may also support a use case where one or more trusted tag creators issue tags linked to previous tags created by another untrusted tag creator, certifying certain tags as trustworthy.

Other users may opt to use another trust model that includes authorities who certify entities that provide information to the registry. This model is commonly used for Internet browsing to distinguish between trusted and untrusted websites. For example, an organization may register a public key with a company like Verisign. That organization can then use its key as a certificate on any websites it chooses to validate its identity and trusted status. Visitors that trust Verisign's certification process can determine whether or

not to trust the website by checking if the public key is registered with Verisign's Certificate Authority. A similar system is possible to determine the validity or quality of SWID tags, *and/or* the software they are tagging.

In recent years, another model has garnered increased attention. This model uses analytics to assess a reputation score based on a number of factors. This score can allow organizations to choose what scores they will require, either to determine if they should trust a tag on the blockchain, or if they should trust a software product registered on the blockchain. This provides great potential for companies to use more advanced analytics such as Monte Carlo analysis to optimize their assessment of trustworthiness based of a much larger range of data.

The availability of these diverse options presents tool developers and organizations with flexibility to customize trust decisions according to their needs.

3.2 Financial Feasibility:

The core benefits of the proposed system are to enable new capabilities, enhance modularity of SAM systems through the use of existing standards, and make processes more economical; however, the design itself can be structured to collect enough income to be self-sustaining. While the blockchain is open to the public, the requirements to be able to add records to the blockchain can and should be restrictive to ensure the health of the blockchain and the quality of the data on it. For example, a software producer could be charged per tag issued. This can be nominal for a software producer compared to the cost of creating or deploying new software or a software patch, but it also protects the chain from being overloaded by fake or low-quality tags. This is a type of Denial of

Service scenario referred to through the rest of this document as “flooding.” In some observed cases, users will sometimes repurpose blockchains from their intended use to store other data in a publicly protected and immutable data repository. The best price structure for the SWID and Ent registries may be to ensure that the cost is higher than other immutable data storage services, such as the Ethereum blockchain (which will store any data a user likes), but be priced low enough to be a negligible expense for software producers and consumers and provide a minimal impediment to adoption of the SWID and Ent registries into their tool sets.

A small fee based on the number of tags or the amount of data used addresses two needs for the system. First, it can help cover the cost of system maintenance, and second, it can discourage mass uploads which could reduce the usefulness and value of the blockchain to its users. If the blockchain grows too large to function on a standard platform, the cumulative sum of small fees could cover the cost of scaling to a big data infrastructure, though this is unlikely to be a constraining factor. We can note that the blockchain for Bitcoin, a highly active cryptocurrency, had not surpassed 20GB within its first five years of operation. While SWID tags and Ent's will typically contain more data than a Bitcoin transaction, the community of tag creators is expected to be much smaller and more targeted, resulting in a lower frequency of transactions. The infrastructure developed in this thesis could scale to store ten 10TB of data—enough to store billions reasonably sized tags.

Chapter 4: Competing Capabilities

This research did not identify any existing capabilities that perform the services identified in the concept proposed in this assessment. That said, there are existing SWID-based capabilities that could be seen to compete indirectly with SWID tag and Ent registries.

TagVault.org is a not-for-profit certification authority for software tagging which focuses on ISO 19770-2 and ISO 19770-3 standards. It provides a tag validation tool both for tag producers to verify that the tag is properly formed, and for tag consumers to validate that a signature of a tag is valid, and that the tag content has not been modified (tagvault.org, 2018). While valuable, these resources do not provide many of the advantages envisioned from the SWID tag and Ent registries, such as allowing configurable levels of trust and signatures for software discovery tools.

Additionally, Steve Clos, the director of TagVault.org, co-founded Managesoft, which developed a database of Software Identification tags for open source tools, and several SAM tools. Flexera Software LLC acquired ManageSoft in 2010 and now controls its assets, including its proprietary database of software tags (Bloomberg). Since this database is used for scanning tools, and not for many of the applications envisioned for the SWID and Ent registries, it is not considered to be in direct competition.

Other competing capabilities do not make use of software tagging at all, but seek to provide some of the same services. One example are heuristic tools. These tools extract available data such as file size, file hash, installed location, program test output, etc. and

use the available information to automatically determine a best guess for the identification of the software. As the science of artificial intelligence and deep learning continue to improve, they may be used to create much more effective programs that could provide similar services as those provided by software tag-based tools. Since many environments will likely have a mix of both tagged and untagged software, heuristic tools may also provide complementary information to provide insight into untagged software in a given environment.

Chapter 5: SWID and Ent Registry Design

This section details the plans for implementation of the registry infrastructure for the concept described above. A Hyperledger Fabric blockchain implementation supports two blockchain ledgers. One contains SWID tag information with the unique SWID Tag ID as a key and the XML formatted text of the tag as the value. The second contains Ent information, with the Ent ID as a key, and the XML-formatted text for the Ent as the value.

5.1 Key Architectural Components

Each of the technologies described below provides crucial resources that are necessary to create a blockchain registry service.

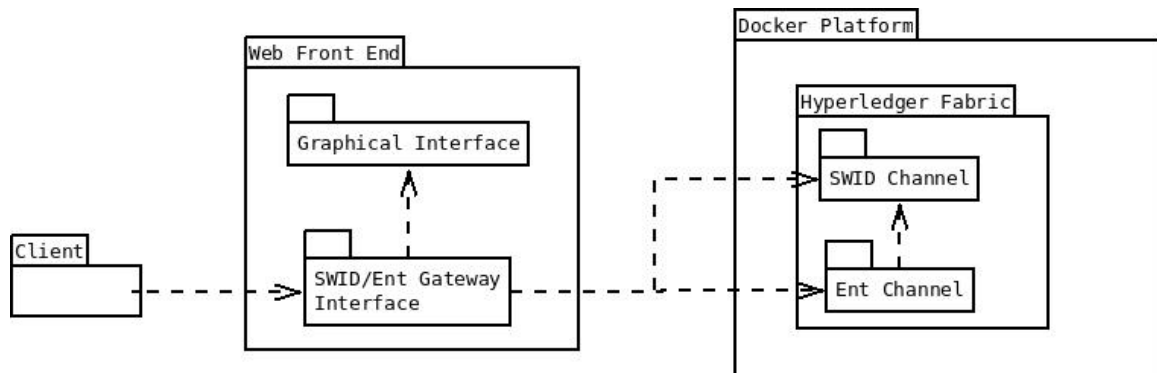


Figure 4. Key Components of SWID and Ent Registries

5.1.1 Cloud Server (IaaS)

A virtual cloud server with a static IP address supports the entire application. In the rest of this section, this virtual server will be referred to as the Cloud Server. The

implementation on a single server is sufficient for a proof of concept, but the it does not have any of the security benefits of a distributed blockchain. A more secure application would need to run on multiple server instances.

5.1.2 Docker Platform

Docker containers have become a leading cloud technology. They allow for highly modular design and can guarantee transportability of applications across platforms. The Hyperledger Fabric blockchain implementation consists of five executable files. However, all instructional documentation centers on the deployment of docker containers to host each of the Hyperledger entities. The Docker platform runs on top of the Cloud Server operating system and access to all Hyperledger Fabric services occurs through these containers.

5.1.3 Blockchain Solution

Hyperledger Fabric was selected as the blockchain solution for the foundational blockchain technology for this application. The rationale for selection of Hyperledger fabric is detailed in Section 5.2. A different smart contract (i.e., chaincode) was added to each of two channels ensuring the proper formatting for additions of SWID tags (first ledger) and Ents (second ledger). The Hyperledger Fabric provides an API to interact with a blockchain platform. The API can be used to add and retrieve records from a blockchain. This includes issuing chaincode, which will be used to define the roles, and the permitted operations for interaction with both the SWID tag and Ent ledgers. The Hyperledger Fabric blockchain supports integration with different databases which have different performance characteristics. At the time of this writing, Hyperledger Fabric

supports both LevelDB and CouchDB databases. (For additional information, see <http://leveldb.org> and <https://couchdb.apache.org>.) This research was conducted on CouchDB, which allows for constant time lookups for information since the underlying data structure is based on a B tree. A B tree is a data structure for storing key-value pairs that allows access to data in constant time for systems with fixed key lengths, regardless of the amount of data the structure is storing. Different channels are able manage permissions for different users; however, this functionality is not necessary for the design of the registry. Standard practices exist to provide signatures within software tags themselves, making digital signing using the Hyperledger Fabric utilities redundant and overly restrictive. Therefore, all invocations and queries to the SWID/Ent ledgers were passed through a single user account, where both the private and public keys of that account were made accessible to all users. This allows for digital signing to be handled externally from the Hyperledger Fabric processes.

5.1.4 SWID/Ent Channel Smart Contract Functions

The smart contracts that can be used for both the SWID tag and Ent ledgers must enforce conformance to the ISO standards 19770-2 and 19770-3 for software identification tags and software entitlements. This ensures proper formatting for every record.

5.1.5 SWID/Ent Gateway Interface

The SWID/Ent gateway interface allows https traffic from the public Internet. This will allow users to interface with the web-based graphical user interface front end.

The https certificate for the web server will not be registered with a certificate authority for this proof of concept.

5.1.6 Web Front End

The web front end provides a web-based graphical user interface, allowing users to perform the use cases described in section 5.2. Namely, it allows users to manually enter and retrieve tag information. In the future, a machine-to-machine interface would much more effectively integrate with automated SAM and security tools, but the graphical user interface is sufficient to demonstrate the concept.

5.2 Selection of Hyperledger Fabric:

At the time of this writing, the Linux Foundation's Hyperledger project has five business blockchain frameworks. Hyperledger Fabric was selected as the foundational technology for the blockchain database since it was the only Hyperledger framework that was approved for deployment in production at the time development began SWID and Ent registries.

Two other possible blockchain implementations that could have filled this role include Hyperledger Sawtooth and Hyperledger Iroha. The Hyperledger project describes Hyperledger Sawtooth as "a modular platform for building, deploying, and running distributed ledgers. Distributed ledgers provide a digital record (such as asset ownership) that is maintained without a central authority or implementation." Some key advantages to this framework are its ability to run both as a permissioned and a permissionless blockchain. A permissioned blockchain means that the entities that participate on the

blockchain must be identified and given permission to interact with the blockchain. A permissionless blockchain allows for decentralized validation where anyone can participate.

The Hyperledger project states that Hyperledger Iroha is “designed to be simple and easy to incorporate into infrastructural projects requires distributed ledger technology. Hyperledger Iroha features a simple construction; modern, domain-driven C++ design, emphasis on mobile application development and a new, chain-based Byzantine Fault Tolerant consensus algorithm, called Sumeragi.” The main advantages of the Iroha blockchain is that it is lightweight and its consensus algorithm can function well in environments with unreliable connectivity, such as mobile devices.

5.3 Hyperledger Fabric Architecture

During the course of the development of application, it became clear that the Hyperledger Fabric blockchain has not yet reached a sufficient level of maturity for this application. Several months of research and testing failed to provide a technical description of the Hyperledger Fabric components, and how they work together. Professional training through a company that contributes code to the Hyperledger Fabric project, Altoros Systems, was able to provide descriptions of the basic components and design decisions of the Hyperledger Fabric system. However, the training made clear that deployment of a production application would require professional consultation from individuals involved in the development of the project, or a well-resourced dedicated team to perform the necessary development operations functions. Some of the relevant information from this training is reconstituted below.

5.3.1 Key Concepts

Blockchain Ledger. Data stored in the form of a list. In the context of Hyperledger Fabric, it includes all permission configurations for entities that are able to interact with the ledger, the smart contracts that can function on the ledger, and all transactions that change data on the ledger. In Hyperledger Fabric, each item in the ledger's list includes a hash of the previous list item, making the ledger into a blockchain.

Smart Contract (also called "chaincode"). A smart contract is a software program that reads or writes state to a ledger. Any state that a smart contract writes to a ledger is called a transaction. In Hyperledger Fabric, peers have a copy of the smart contracts and a client can initiate its execution by requesting that peers execute the smart contract and provide any data inputs the peers require to run smart contract.

Transaction. Any record proposed to be added, or previously added to a ledger following the successful invocation of a smart contract. A client can propose a transaction, and if the invocation is successful, the transaction is said to be valid. If the invocation is unsuccessful, the transaction is said to be invalid.

Channel. A blockchain ledger. Peers can participate on multiple channels, meaning that they are reading and/or operating on multiple blockchain ledgers.

Endorser. Endorsers determine whether proposed transactions are valid or invalid. If a client obtains the appropriate endorsements determined by a smart contract's endorsement policy, the client can send the transaction with the appropriate endorsements to the ordering service, which can validate the endorsements and commit the transaction to the ledger. Transactions that fail to obtain the appropriate endorsements will not be added to the ledger.

5.3.2 Types of Nodes

Clients. Submits transactions to the peers and to the orderer

Peers. Maintains a copy of the ledger and adds any new valid transactions to its ledger. Some peers function as endorsers.

Ordering Service. Ensures consistent sequencing of transactions and delivers all valid transactions to all subscribed peers

5.3.3 Transaction Time Sequence Diagram

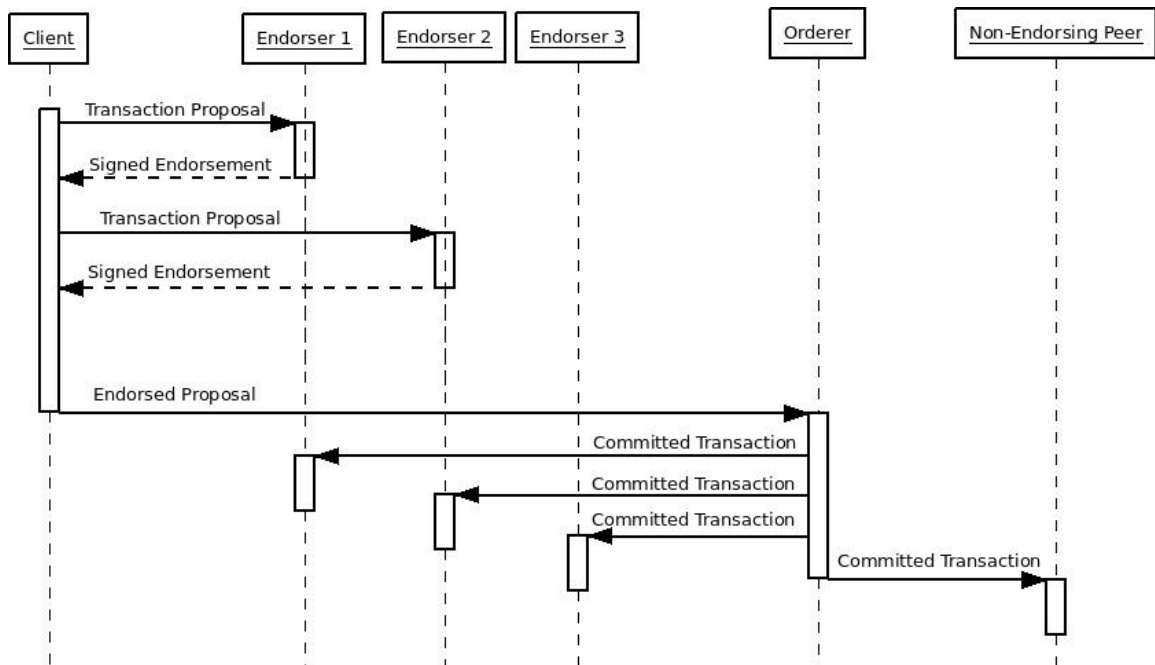


Figure 5. Sequence of Successful Hyperledger Fabric Transaction

The image above illustrates an example sequence for a successful transaction.

1) The client sends a transaction proposal to an endorsing peer (Endorser 1), specifying a method it would like to invoke on the smart contract, and supplying any additional information required for the transaction. The peer processes the proposal by running the specified method in the smart contract to ensure that the transaction runs

successfully within the parameters specified by the smart contract. Once the transaction is processed successfully, the endorser signs the transaction and provides the signature back to the client.

2) The client sends the same transaction proposal to a second endorsing peer (Endorser 2). This can take place concurrently with the previous step. The second endorsing peer also checks that the transaction operates within the parameters specified by the smart contract, signs the transaction proposal, and sends the signature back to the client.

3) If an endorsement policy for the smart contract does not require endorsements from additional peers, like Endorser 3, the client will not need to seek additional endorsements.

4) The client attaches the endorsement signatures to the transaction proposal and sends it to the orderer.

5) The orderer checks the endorsement policy of the invoked smart contract. In this case, signatures are required for Endorser 1 and Endorser 2. The orderer validates that the tag has the two required signatures, assigns a unique block number to the transaction, and signs the transaction proposal and endorsements, and the transaction proposal is then considered a transaction. The transaction, combined with the endorsement data is called a block.

6) The orderer sends the block to all subscribed peers. This includes any endorsers for the smart contract, any endorsers that are not required to sign the smart contract, and any peers that are subscribed to the blockchain, but do not provide endorsements.

5.4 Roles and Use Cases

This section addresses three sets of use cases. The first section (section 5.4.1) addresses the use cases that have been fully implemented over the course of this research. The second section (section 5.4.2) addresses use cases that were planned, but were not implemented since their implementation would not have been informative.

5.4.1 Supported Roles and Use Cases

The section below outlines the basic roles and use cases addressed by the system developed.

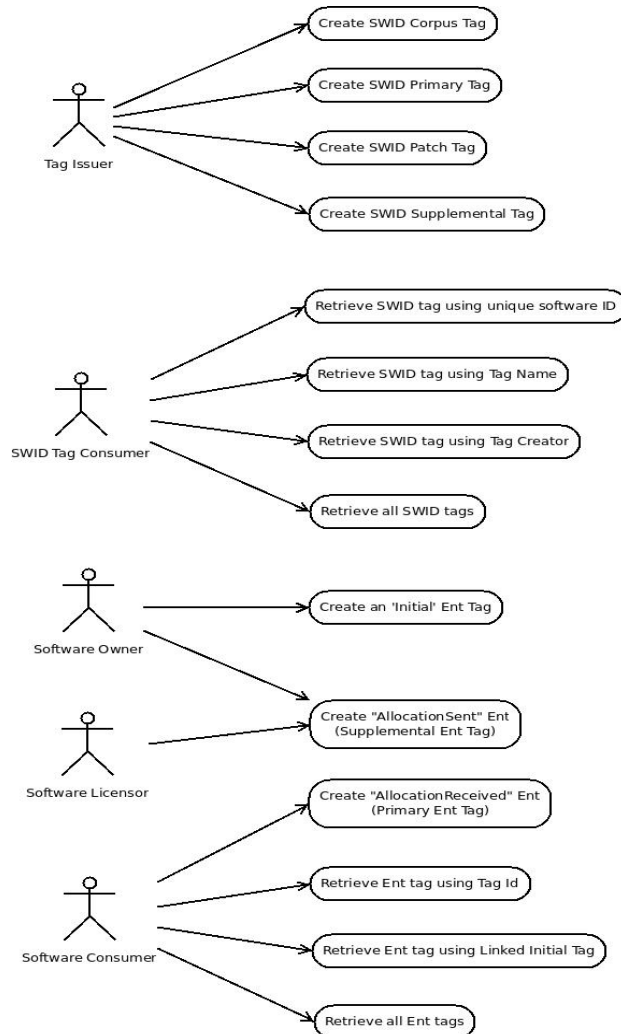


Figure 6. Implemented Use Cases

1. SWID Tag Issuer Creates a Corpus SWID Tag: A Tag Issuer can add an entry to the SWID blockchain for a new software product. Per ISO Standard 19770-2, Corpus Tags are intended to tag pre-installation distributions of software.
2. SWID Tag Issuer Creates a Primary SWID Tag: A Tag Issuer can add an entry to the SWID blockchain for a software product or a bundle of software products.
3. SWID Tag Issuer Creates a Patch SWID Tag: A Tag Issuer can add an entry to the SWID blockchain for a patch which is linked to one or more existing products and any other patches to which it may have a relation.
4. SWID Tag Issuer Creates a Supplemental SWID Tag: A Tag Issuer can add an entry to the SWID blockchain for a tag providing additional information about an existing software product. Per ISO Standard 19770-2, all Supplemental SWID tags must have a link to a Primary SWID tag.
5. SWID Tag Consumer Retrieves a SWID Tag Using a Unique Software ID: A SWID tag Consumer can submit a unique Software ID to retrieve SWID tag information on that product. Unique Software IDs will conform the tag identifier as defined in ISO Standard 19770-2. The SWID tag ID servers as a key, and the SWID tag itself is the value of a key-value pair stored on the blockchain, allowing a utility to query the blockchain for the appropriate record.
6. SWID Tag Consumer Retrieves SWID Tags Using the Tag Name: SWID tag Consumers can submit a tag Name to retrieve all tags bearing that tag Name. ISO 19770-2 requires all SWID tags to have a tag Name, so this query can be used to retrieve any tags. Tag Names are not required to be unique, so they can be used to retrieve multiple tags.

7. SWID Tag Consumer Retrieves SWID Tags Using the Name of the Creating Organization: A SWID tag Consumer can submit the name of an organization and retrieve a list of all of the tags created by that organization.
8. SWID Tag Consumer Retrieves all SWID Tags: A SWID tag Consumer can issue a request to retrieve a list of all SWID tags.
9. Software Owner Creates 'Initial' Ent Tag: The Software Owner can create an initial Ent which associates a Software Owner to a specific software product (i.e., associates a software owner to a SWID tag).
10. Software Owner/Software Licensor Creates an 'AllocationSent' Ent: Both Software Owners and authorized Licensors can allocate licenses to software licensees (most commonly Software Consumers). This adds a record to the blockchain noting that the allocation has been made.
11. Software Consumer Creates an 'AllocationReceived' Ent Tag: A Software Consumer can add an 'AllocationReceived' Ent tag to the blockchain. This responds to an 'AllocationSent' Ent Tag issued by a Software Licensor.
12. Ent Tag Consumer Retrieves an Ent Tag Using a Unique Ent Tag ID: An Ent Consumer can submit a unique Ent ID to retrieve the related Ent information. Unique software IDs must conform to the tag Identifier as defined in ISO Standard 19770-3. The Ent ID servers as a key, and the Ent itself is the value of a key-value pair stored on the blockchain, allowing a utility to query the blockchain for the appropriate record.
13. Ent Tag Consumer Retrieves Ent Tags Using the Tag ID of its Linked Initial Tag: An Ent Consumer can submit the Ent ID of an Initial Ent and retrieve the initial

Ent and a list of all of the Supplemental Ents that are linked to that Initial Ent.

Results will include one Initial Ent and a list of zero or more supplemental Ents linked to that initial Ent.

14. Ent Tag Consumer Retrieves All Ent Tags: An Ent consumer can issue a request to retrieve a list of all Ents.

5.4.2 Unsupported 19770-3 Use Cases

Following the creation of classes to validate all entities for the SWID standard, implementing full functionality to validate information for the Ent would have required a nearly identical process. The process would have required a tailored implementation since Hyperledger Fabric smart contracts are written in Golang, and there is currently no credible software that could validate a tag with the XML schema provided by the ISO. While validation of data using an XML schema will not currently work within the Hyperledger Framework, validation of XML schema is a tried-and-true technology, and recreating this functionality for Ents did not further the research objectives. The use cases below describe use cases that should have been implemented per the 19770 Standard, but that were not due to this limitation. The new use cases that were included have been added to the Use Case diagram in red with a gray background below.

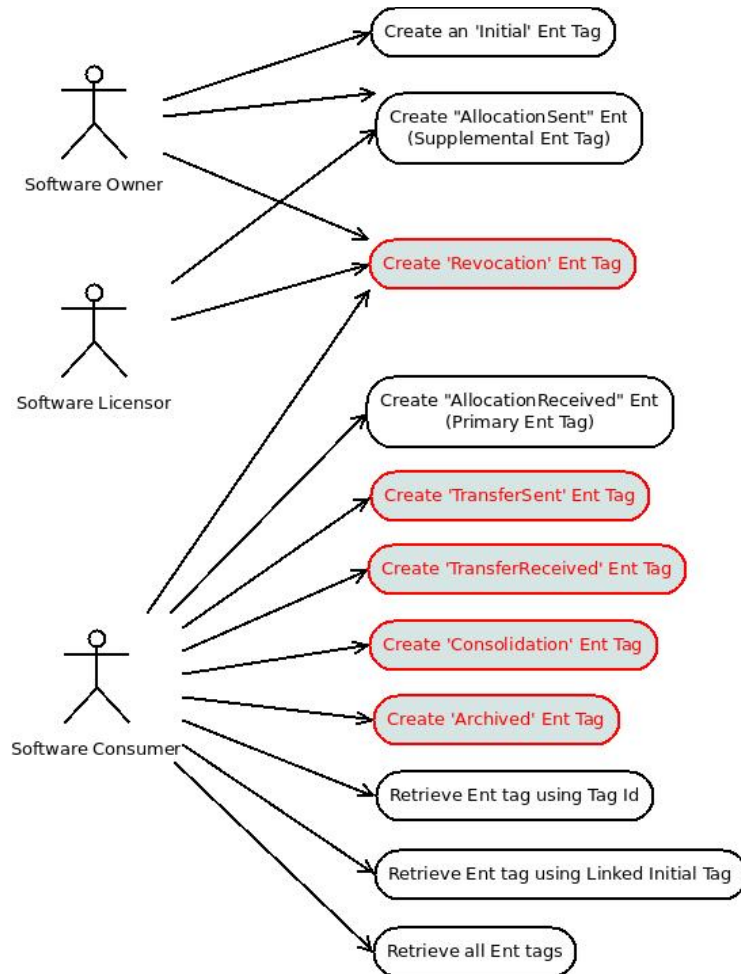


Figure 7. Supported and Unsupported ISO Standard 19770-3 Use Cases

1. Software Licensor/Software Consumer creates ‘Revocation’ Ent: Software Licensors and Software Consumers can add an Ent to the blockchain revoking an existing Ent. This may be done for example if there was an error made in the initial Ent.
2. Software Consumer Creates a ‘TransferSent’ Ent: Software Consumers can reallocate their software assets. For example, a corporate headquarters may wish to transfer software entitlements to one of its branches, or during a merger an acquired organization’s entitlement may be consolidated into the acquiring

organization. By adding a 'TransferSent' Ent to the blockchain, the software entitlements associated to one owner can be shifted to another.

3. Software Consumer Creates a 'TransferReceived' Ent: In response to the issuance of a 'TransferSent' Ent, the designated recipient can receive the transfer by adding a 'TransferReceived' Ent to the blockchain.
4. Software Consumer Creates a 'Consolidation' Ent: In accordance with ISO Standard 19770-3, a 'Consolidation' Ent can be used to combine multiple different entitlements into a single entitlement.
5. Software Consumer Creates a 'Archived' Ent: While not necessary on a blockchain record, to comply with ISO Standard 19770-3 and integrate with any tools compliant with that Standard, Organizations can issue an Ent to the blockchain denoting certain Ents as 'archived'.

Chapter 6: SWID and Ent Registry Development

All preliminary research on blockchain technologies and the 19770 Standards was completed in July 2017. Development on the research application began in late July. This section describes the stages of development.

6.1 Implementation Process

The initial development plan for the components detailed in Section 5.1 started with the services with the fewest dependencies on other parts of the system, and progressed to those with the most dependencies. Setup of the initial environment was not dependent on any aspects of the application so it was addressed first. All other elements relied, either directly or indirectly on the Hyperledger Fabric network. This step was the most time-consuming and revealed the greatest challenges for using Hyperledger Fabric for the creation of SWID tag and Ent registries, and an adequate network configuration was never achieved. Use of a template network was sufficient to test smart contracts for the SWID and Ent blockchains so work could progress. The SWID and Ent blockchains are independent of each other, and so could be implemented in any order. The SWID blockchain was selected first since it requires simpler operations. Ent blockchain could use same functionality as the SWID blockchain and extend it to address additional needs.

When the ledgers were completed, the Gateway Interface was developed to retrieve and commit data to the blockchains and a graphical user interface was created to enable users to interact with the Gateway Interface to retrieve and commit data.

The development steps proceeded as follows:

1. Environment setup
2. Set up the Hyperledger Fabric network
3. Develop smart contracts for the SWID blockchain
4. Develop smart contracts for the Ent blockchain
5. Develop the Gateway Interface
6. Develop the graphical website for the user-facing application
7. Deploy application to cloud server

6.1.1 Environment Setup

The environment consisted of a virtual machine run on a VirtualBox hypervisor. The virtual machine was used to host a minimal Ubuntu 16.04 operating system. Golang, Docker, and NodeJS resources were downloaded in accordance with the the Hyperledger documentation for needed prerequisites (Hyperledger, 2017). Setup of these resources was trivial and will not be detailed further.

6.1.2 Hyperledger Fabric Network

The initial goal for the Hyperledger Fabric network was to store a key-value pair and return the appropriate value when a query provided the appropriate key. Substantial research failed to uncover documentation or methods providing guidance for how to implement this functionality with acceptable levels of security, reliability, and flexibility. Following several months of investigation and testing, efforts to establish a suitable Hyperledger Fabric network were discontinued. Some challenges that led to the abandonment of a suitable network configuration are detailed in section 6.2.

In lieu of a network designed for the SWID and Ent blockchain registries, a sample single-peer, single-channel network configuration provided by the Hyperledger project served as a template. The sample configurations could not be used to realize any

security, reliability, or transparency benefits expected from a blockchain implementation. Attempts to modify the sample to incorporate additional peers or support multiple channels were unsuccessful.

During work, maintaining versions using built-in scripts often specified paths to remotely hosted scripts and containers that negatively impacted efforts for version control for the containers used for the application. Since updates are often not backward-compatible, these hidden updates disrupted development several times. Best practice dictates that versions are selected and updates are only deployed when their effectiveness can be validated through a change management procedure. It is also not clear how to retrieve older versions of scripts once they had been altered by other scripts, so frequent restorations from backup files were necessary.

While some of these issues required workarounds for the development of other elements of the system, using an existing sample network as a template was sufficient to allow for testing of smart contracts for the SWID and Ent blockchains.

6.1.1 Smart Contract for the SWID Blockchain

Early in the development of the SWID blockchain, it became clear that development would need to take place in two phases. A healthy rhythm for coding includes a tight feedback loop from adding code to seeing the results of tests validating the new code that has been created. Testing includes both ensuring that the code can compile and that it can run and pass all tests. In an effective development environment, each iteration of this loop should only take a few seconds. When developing within the Hyperledger Fabric network, tests between code changes required removal of a container,

and initialization of new container which took between 90 and 110 seconds. This extended the feedback loop to roughly two minutes. Therefore, functionality for the smart contract was first developed separately from the Hyperledger Fabric network on a separate test server where it could be quickly tested. Once the program worked, it was ported over to the Hyperledger Fabric network where troubleshooting the integration of the smart contract could take place.

Hyperledger Fabric currently has an API that allows smart contracts to be written in Golang or Java. All early development work had been conducted in Golang, so Golang was selected for implementation of both the SWID and Ent smart contracts. Part way through development, it became clear that the publicly available Golang libraries would not be adequate for the implementation of the 19770-2 or 19770-3 standards. Golang has a standard library for marshaling and unmarshaling XML and JSON data. Unfortunately these functions require the object model to be completely defined in order to extract the necessary data. Both the 19770-2 and the 19770-3 ISO standards are infinitely extensible. Any tag creator can define its own objects or attributes and include them in a tag. In these cases, the available Golang libraries would ignore these data. This means that to use Golang, a custom parser would be required to validate that each tag submitted complied with ISO Standards, and extract the data into memory.

Utilities are available for many languages that take input from the XML Schema Definition (XSD) file and validate conformance to the file, and other tools can automate the construction of a data model using the conditions codified in an XSD file. While some of these tools exist for Golang, none of them have achieved the level of

completeness or reliability to be used in production code. The object model used in the smart contract was coded manually using the Golang XML and JSON encoding libraries.

The smart contract was successfully developed to ensure the proper usage of all XML elements described in the ISO 19770-2 Standard, but it did not permit users to include custom objects or attributes. This means that the implementation using standard Golang libraries was not successful in fully implementing the the ISO 19770-2 Standard, and additional work, or a new approach, would be required.

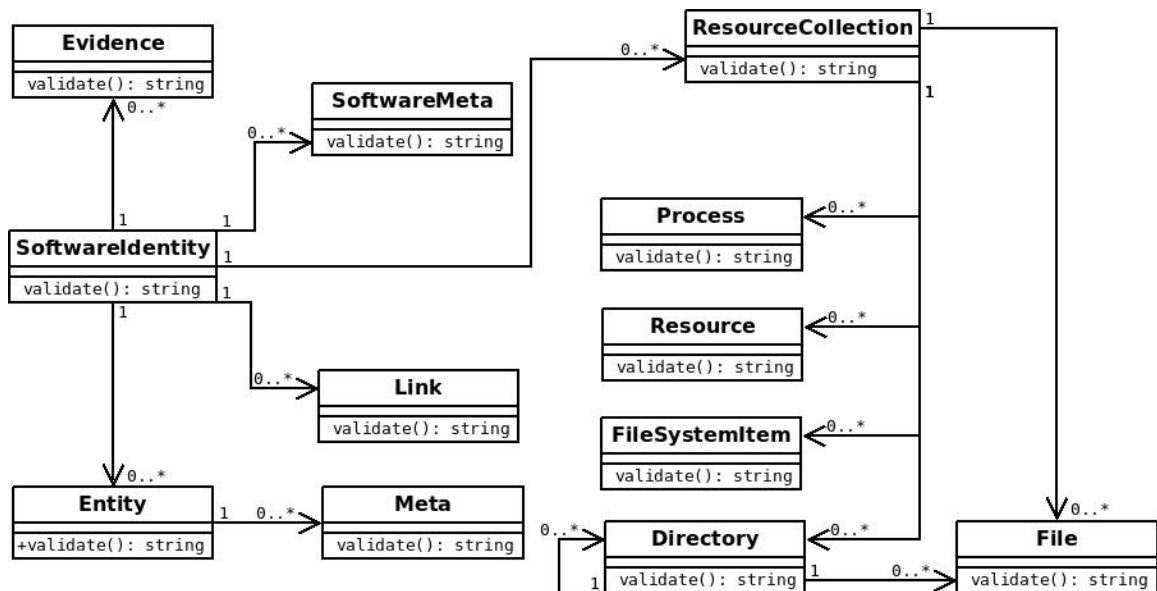


Figure 8. Pseudo-UML Diagram of Validator Design

The primary purpose of the smart contract is to ensure that tags added to the SWID database are all compliant with the 19770-2 standard. The smart contract makes use of a pattern similar to a composite design pattern to ensure that validation can check all elements directly or indirectly contained within the top level SoftwareIdentity object. A composite design pattern was not used since the Golang XML and json encoder libraries require all attributes of child elements to be visible to their parent elements.

Since the separation of concerns was already broken, establishment of a common interface would not have been helpful.

Each defined type within the software identity data model has a “validate()” method which ensures that every attribute for that type is conformant with the ISO 19770-2 Standard. Each parent object then runs the validate() method on all of its child objects if there are any. This creates a recursive validation call that ensures every data field is compliant with the standard. This validation must pass for all objects before the smart contract will permit a new tag to be committed to the blockchain.

6.1.2 Smart Contract for the Ent Blockchain

The design of the smart contract for adding Ents is closely related to the design of the smart contract used for SWID tags. Since the process would be similar to develop a contract to validate all of the objects and attributes for the Ent Standard as it was for the SWID standard, it would not have been informative to manually recreate that functionality. In future versions, this service should be replaced by a reliable library that could validate conformance of each tag with the XML Schema Definition provided by the ISO. However, as mentioned above, research did not identify a Golang library that would be adequate for a production application.

One notable requirement for Ents that differed from SWID tags was a need to query existing data contained within the blockchain in order to determine if a new tag is valid. Specifically, entitlements in ISO Standard 19770-3 contain a “quantification” attribute which may express, for example, how many instances of a software a company

is entitled to use. If that organization would like to reallocate or transfer more entitlements than they have rights to, the transaction should be invalid.

The Hyperledger API provides functionality to retrieve existing state stored within the ledger. The application makes use of these functions to read all additions and subtractions to the quantification of a software entitlement from all linked tags. The existence of sufficient entitlements is required for a transaction to be considered valid.

6.1.3 Gateway Interface

The Gateway Interface is an ExpressJS server that responds to HTTP requests. It responds to GET requests for the home page by providing HTML files that make up the graphical web-based interface. When the user issues commands through the graphical interface, the Gateway Interface will route the resulting POST request to the appropriate smart contract and method. This routing is performed by calls to the Hyperledger Fabric API based on NodeJS. Methods from this API are used for the submission and retrieval of all information to the Hyperledger Fabric ledger.

6.1.4 Graphical Web-Based Interface

The graphical user interface consists of five HTML files for each of the five following functions:

1. Introductory page
2. Submission of SWID tag
3. Query SWID tags
4. Submission of Ent
5. Query Ents

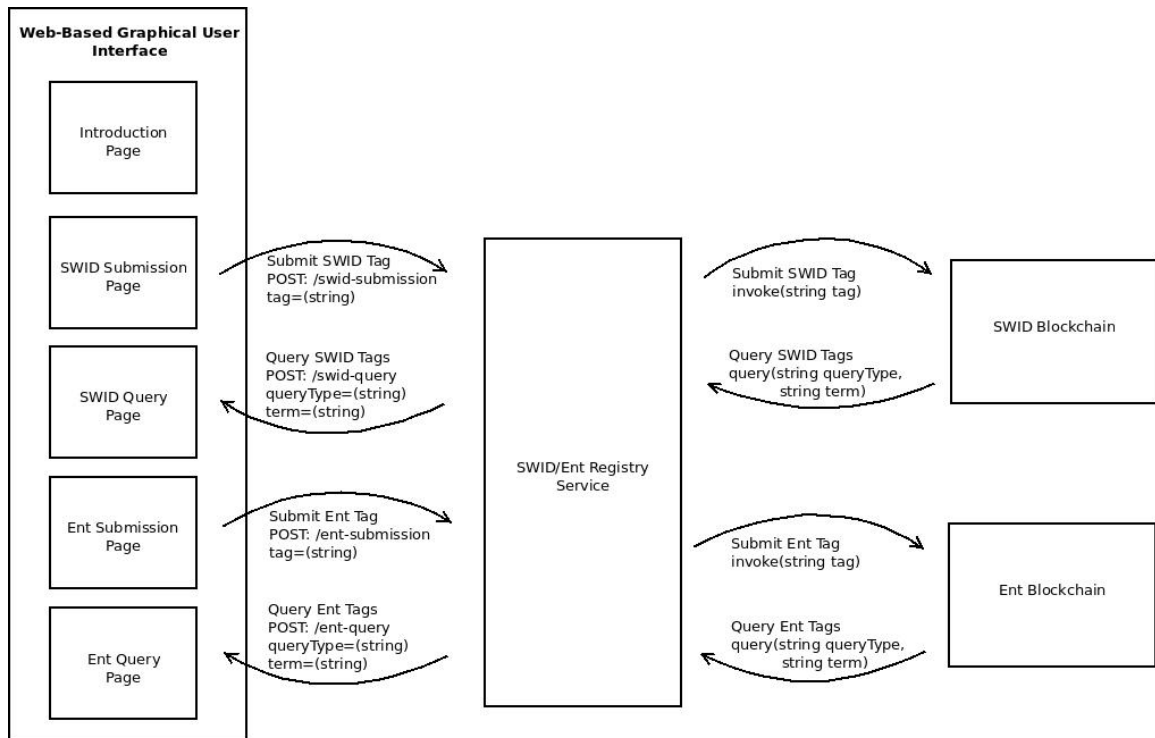


Figure 9. Graphical User Interface Page Functions

Since pages are very small, all data is preloaded upon visiting the homepage to ensure fast navigation. All data returned following submission attempts or queries is returned through asynchronous calls to limit delay due to data transmission over network connections.

The Introductory page presents summary text about the application and a navigation menu to the other pages.

The pages for SWID tag and Ent submission contain a text area for the user to enter a tag and a submit button to send the request to commit the entered contract to the blockchain for validation. If the request fails, a Javascript alert provides details about the error. If a request succeeds, a Javascript alert will provide a notification that the request has succeeded and provide the ID number of the new block containing the tag that was submitted.

The pages for SWID tag and Ent retrieval contain a dropdown used to select the field the user would like to use to query the tag. A text box is provided for users to enter the key they would like to query on. A submit button sends an asynchronous request to the server to search the index specified in the dropdown for the key specified in the text box. If there are no results to return, the area below the search parameters will appear empty. If there are results, they will be printed in a list below the search parameters.

6.1.5 Deployment to Cloud

Deployment to the cloud consisted of the steps listed in section 6.1.1. Following the necessary setup of the environment, all files in the VM development environment were copied to the Cloud server. The ExpressJS server was modified to reply to requests on the public Internet. Launching the Gateway Interface and booting the Hyperledger Fabric network completed the system for the proof of concept.

6.2 Review of Hyperledger Fabric Implementation

Hyperledger Fabric was a crucial component in the development of this proof of concept. Several features of the Hyperledger Fabric have the potential to provide useful capabilities for the applications, but unfortunately it was found to be unsuitable for a registry for software identification and tracking purposes as planned in this document. The following two subsections address both the strengths and weaknesses of the Hyperledger Fabric discovered during the development of this proof of concept.

6.2.1 Hyperledger Strengths

Concept of channels which allows clean separation of ledgers: Channels permit a Hyperledger network to separate data into separate ledgers. In the case of the SWID and Ent registry, this would permit greater modularity in the design where one ledger could hold all SWID tags and one ledger could hold all Ent's. There are several advantages to this division. First, if a smart contract is vulnerable to exploitation, it cannot write directly to another ledger, so separate channels may remain unaffected by the breach. Second, channels allow different peers to subscribe to different channels, so peers need only interact with data that is relevant to them in a publish-subscribe type of model. Third, system architectures can maintain better separation of concerns between different types of data. This is advantageous if, for example, one blockchain needs to be replaced by a new implementation, it may only be necessary to change one channel and leave all other functioning channels alone, thereby saving developer resources.

Support for cross-channel querying: Channels are separate ledgers. Participants in a Hyperledger network might participate on several channels, but not on others. A smart contract on one channel cannot directly change the state or activate a smart contract on another channel, but it can query another channel, and use state from that channel in part of the process of its own smart contracts. This allows for blockchains to interleave in many unique ways. For example, if an Ent on one channel were to reference a SWID tag on another channel, it could retrieve the relevant data using the Hyperledger Fabric API and ensure atomicity between the query and the full execution of the smart contract. This could also be useful to track monetary exchanges. For example, if one ledger records

payment information, other ledgers could ensure that funds have transferred successfully before allowing their transactions to succeed.

Messaging to external parties: Hyperledger Fabric allows clients to create event listeners that will notify them of specific events. The most obvious use case for these messages is for a client to listen for a transaction that it has submitted so it can be notified when or if the transaction succeeds. This permits the client to wait for a response rather than query a peer at intervals to see if a transaction has been committed. However, over the course of these investigations, it became clear that these event listeners could be used for more powerful purposes, such as writing data across-channels. Hyperledger Fabric permits read-only queries across channels, but it does not permit a smart contract from one channel to write data to another channel. Event listeners can integrate with simple middleware implementations to provide an efficient means to perform write operations across channels. A client can listen for an event from a smart contract on one ledger, and when the event is triggered, it can use data from that event to execute a different smart contract on a different ledger. Using this method, Hyperledger Fabric can effectively automate interactions between multiple blockchains.

Constant-time record lookups: Hyperledger Fabric permits the use of either LevelDB or CouchDB for its database implementation. The SWID and Ent blockchain used CouchDB which is based on a B tree data structure. The Hyperledger Fabric API also permits an arbitrary number of indexes. Each of these indexes contains an array of strings which is used to hierarchically drill down to the appropriate value. Since CouchDB uses a B tree structure, keys that are made from an equal length will be found in $O(1)$ time. This is useful for data retrieval, especially as ledgers scale to greater sizes.

6.2.2 Hyperledger Weaknesses

Ordering Service is a single point of failure: Blockchain implementations are often touted as data stores that have no single point of failure. Hyperledger Fabric can be designed to decentralize endorsements for its smart contracts, but its ordering service, which determines the sequence that transactions are added to the blockchain and disseminates approved transactions to any subscribed peers, still presents a single point of failure. The ordering service can be made more robust by shifting from a single node ordering service to a cluster of 3-7 nodes functioning in an Apache Kafka messaging fabric. (For additional information, see <https://kafka.apache.org>.) While this cluster can ensure redundancy so that node failures can't bring down the network, the orderer would still be controlled by a single organization. A breach or malicious act by that organization could compromise the blockchain.

Endorsement policies can't be updated once started: Hyperledger fabric stores the keys of all endorsing nodes and the orderer in the first block of the ledger, known as the genesis block. Once the chain is in operation, no new endorsers can be added. This means that all endorsing participants on the blockchain must be identified prior to operation of the ledger, and, if the set of endorsers needs to be modified, a new ledger must be created and integrated into any existing dependent infrastructures.

SAM system as designed only uses one endorser: Hyperledger fabric is well designed to serve many endorsers, however the implementation for the SWID tag and Ent registries only requires one endorser. This introduces another single point of failure. This is not a weakness of the Hyperledger Fabric implementation, but an example of a drawback for the use of Hyperledger Fabric for an Ent and SWID registry.

Documentation Gaps: Throughout the research process, a frequent and severe lack of documentation prevented the set up and management of a Hyperledger Fabric network. The Hyperledger project maintains a site with instructional documentation that describes how to perform certain functions, such as setting up a network, running chaincode, and developing applications that integrate with the blockchain network. A number of sample configurations are provided to demonstrate each of these functions. This effort is useful, but major gaps remain. For example, in a number of the network configurations, certain functionality was said to be “built in” to certain docker containers. This facilitated the process of setting up the example applications, but left gaps for setting up a new network with the configurations that made sense for other applications. In another example, the documentation deferred configurations to an application developer that knows how to build Hyperledger Fabric applications. This suggests that for certain operations, it is expected that many configurations are not feasible without costly consultation from experts. Given the lack of documentation to build this expertise, these consultants would likely need to be drawn from the Hyperledger Fabric development communities. Additionally, no explanation is provided for how the entities within Hyperledger Fabric networks function together to provide useful services for software applications. Since Hyperledger Fabric is very new, there are no books available to fill any information gaps and scant information is available through other online sources.

Insecure defaults: In many online support forums, users are encouraged to use the examples provided by the Hyperledger project, and adapt them to meet different needs. Since insufficient documentation was available to set up a network with a configuration desired for the SWID and Ent blockchains, the SWID and Ent blockchains

use this approach. However, based on several insecure default configurations, applications made in this way should not be used in an application without review by an expert specializing in Hyperledger Fabric networks. One example of an insecure configuration in the sample implementation is that all keys, public and private, are shared by all peers on the network. This means that any peer could impersonate any other peers, and is a highly insecure default configuration that was never mentioned in the documentation. This configuration makes every peer a single point of failure for the system, negating any security advantages for the use of a blockchain solution.

Chapter 7: System Walkthrough

The primary interaction between the SWID and Ent blockchain is expected to be machine-to-machine communication. However, a web-based graphical user interface to manually submit and query tags on the blockchain can provide a more intuitive demonstration of the capability. Outwardly, the system resembles a simple database system that accepts and stores well formed values, and allows those values to be retrieved based on desired search parameters. This section walks through this simple process showing each of the capabilities, and explaining the more complex process taking place underneath.

The screenshots shown below were from an Internet application accessible from a public IP address. The introductory page renders explanatory text, and several buttons allow users to access dedicated pages to perform four basic operations: submit a SWID tag, query for SWID tags, submit an Ent, and query for Ent.

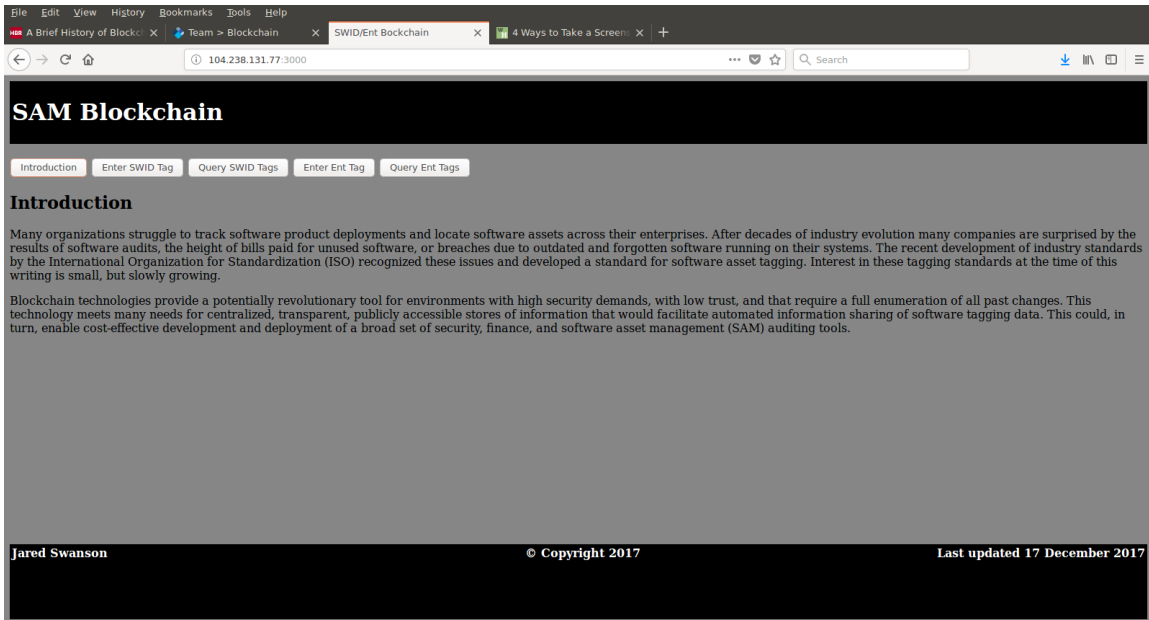


Figure 10. Graphical User Interface Home Page

On the page for SWID tag submission, a user can enter text into a text box. When the user clicks ‘Submit,’ the client sends a POST request to the SWID/Ent gateway service with the user-entered content.

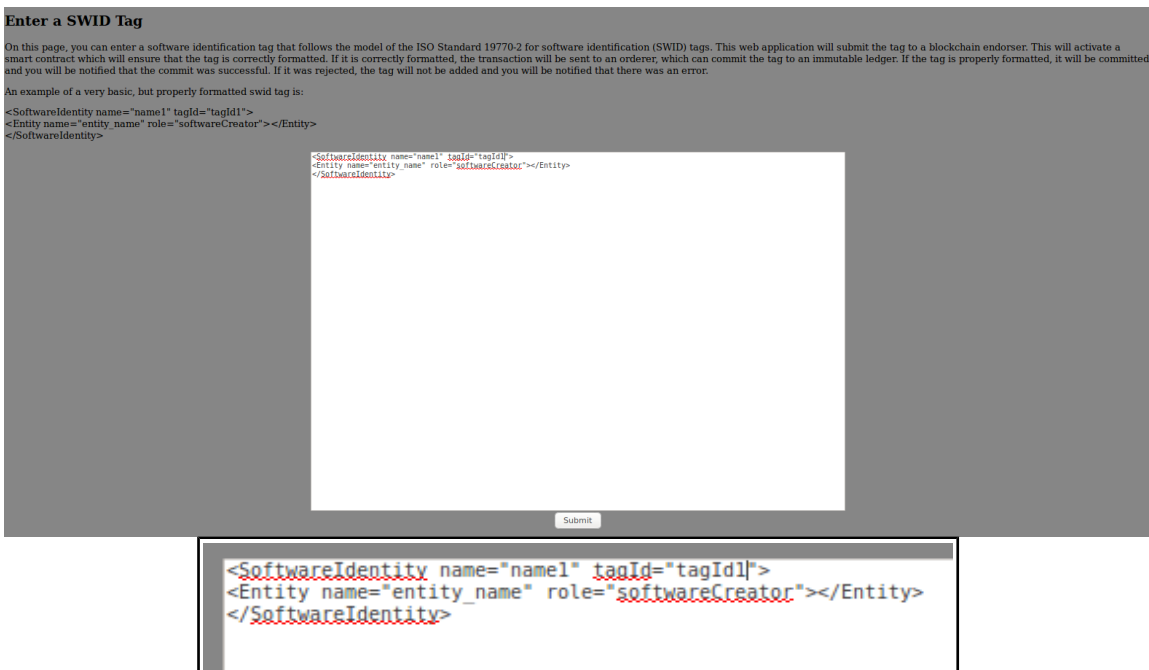


Figure 11. Graphical User Interface SWID Submission

To the user, a successful invocation appears as a successful load into a database, but the submission process is more complex.

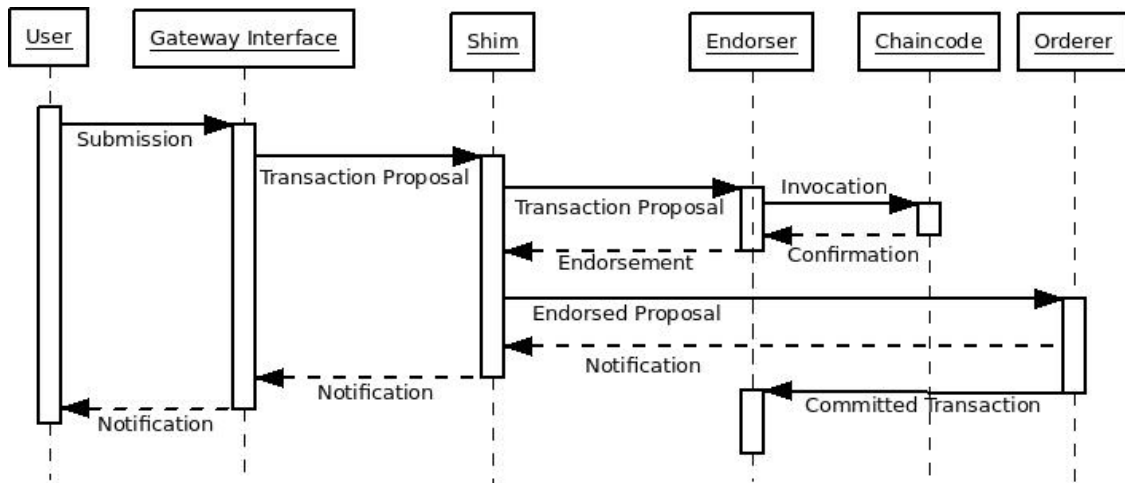


Figure 12. SWID Tag Submission Sequence Diagram

The request content is then passed through a NodeJS API shim, which routes the submission to the Hyperledger peer, and designates which smart contract, and which function on the smart contract the data should be passed to. The smart contract executes on a Docker container separate from the peer. If the function on the smart contract finds that the tag is well formed, the peer will endorse the transaction. Since this proof of concept has only one endorser, only one endorsement is needed and the shim can then pass the transaction with the endorsement signature to the orderer. The orderer can validate that the transaction has the proper endorsements and sign the transaction itself. At this point, the transaction becomes a block. The block is then transmitted to any peers on the network, which will then re-validate that the block has received all necessary endorsements (in this case, it need only validate its own earlier endorsement), and the signature of the order. Once validated, the peer will add the block to its ledger. The shim then returns a message to the SWID/Ent Gateway Interface, signaling the successful

submission of the transaction. This success message is passed back to the client along with the ID of the new block which is displayed to the user, as shown below.

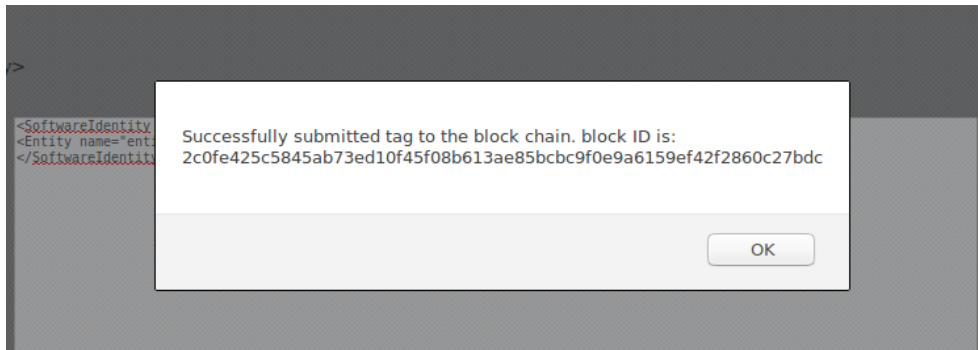


Figure 13. Successful Tag Submission

In the case the submission is unsuccessful, the transaction will never be endorsed by the peer, passed to the orderer, or committed to the ledger. Instead, when the smart contract executes and finds that the transaction is malformed it will pass an error message back to the shim, which will report the error directly to the Gateway Interface. The Gateway Interface then returns the error message to the client, which then alerts the user of the submission error.

For example, if the user submits a new tag as shown above, but leaves out the final closing guillemet, the smart contract will identify the XML syntax error and return details of the error to help identify the reason for the error. The client can then alert the user with an informative error message as shown below.

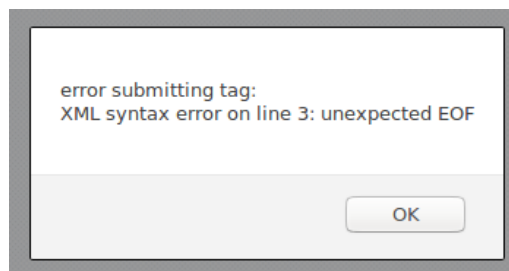


Figure 14. Informative Error

The process to query the blockchain for SWID tags begins similar to the process for SWID tag submissions. The user selects an index to search from the dropdown menu, or requests to retrieve all tags. The user can then enter a key in that index into the textbox and click submit. The index and the search term are both passed to the ExpressJS server in a POST request. Here, instead of calling on the shim to submit the transaction, it will call on the shim to query the transaction. The shim still submits this transaction to the peer to execute the smart contract and, if the transaction is valid, endorse the transaction, but the shim will not submit an endorsed transaction to the orderer. This prevents transactions for queries from being added to the ledger.

If the peer executing the smart contract finds that the query is malformed and an error is returned, the user will be alerted. Otherwise, if the tags are found, the records will be returned and ultimately shown to the user as shown below.

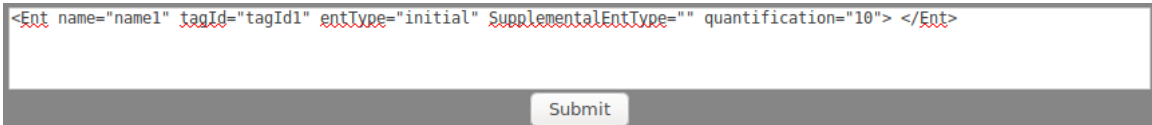


Figure 15. SWID Tag Query Page

Ents use a different smart contract tailored to Ent submissions and queries, but the general process is almost identical to SWID tag submissions and queries, so they will not be covered in detail here. One notable difference, however, is that past transactions in

some cases determine the validity of current transactions. For example, if one tag is entered specifying that an organization has rights to 10 copies of software, that organization can't transfer entitlements to 15 copies of the software to another organization. This is shown another way in the eight frames shown below.

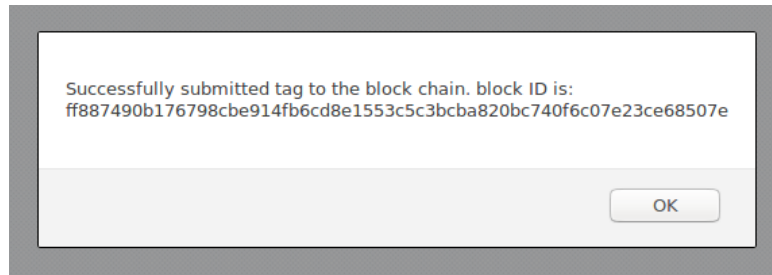
1) An initial Ent is submitted, signifying the rights to 10 copies of software.



The screenshot shows a text input field containing the XML tag: `<Ent name="name1" tagId="tagId1" entType="initial" supplementalEntType="" quantification="10"> </Ent>`. Below the input field is a "Submit" button.

Figure 16. Initial Ent Tag

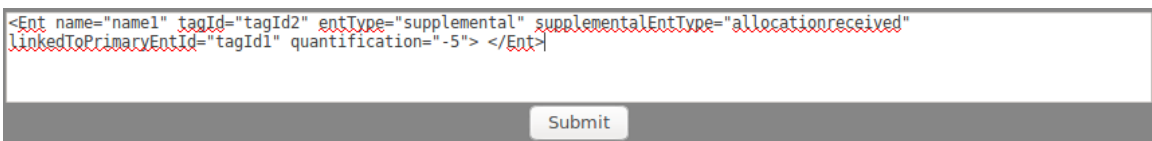
2) The Ent is well formed and the transaction is successful. (Note that the examples used for Ent are not compliant with the standard as discussed in section 6.4.)



The screenshot shows a confirmation dialog box with the text: "Successfully submitted tag to the block chain. block ID is: ff887490b176798cbe914fb6cd8e1553c5c3bcba820bc740f6c07e23ce68507e". There is an "OK" button at the bottom right.

Figure 17. Ent Submission Success 1

3) Another Ent is submitted, this time allocating -5 entitlements, reducing the amount of entitlements available, leaving a total of five.



The screenshot shows a text input field containing the XML tag: `<Ent name="name1" tagId="tagId2" entType="supplemental" supplementalEntType="allocationreceived" linkedToPrimaryEntId="tagId1" quantification="-5"> </Ent>`. Below the input field is a "Submit" button.

Figure 18. Ent Decrementing Entitlements 1

4) The operation succeeds.

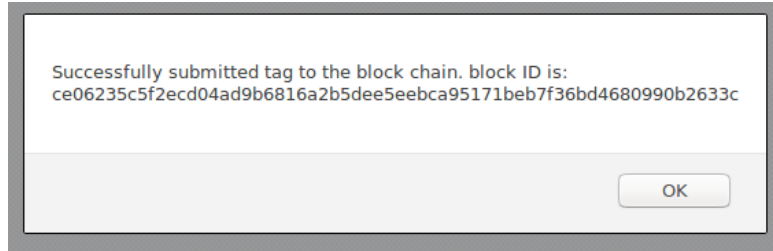


Figure 19. Ent Submission Success 2

5) This process can be repeated, allocating another -5 entitlements since there are enough remaining.

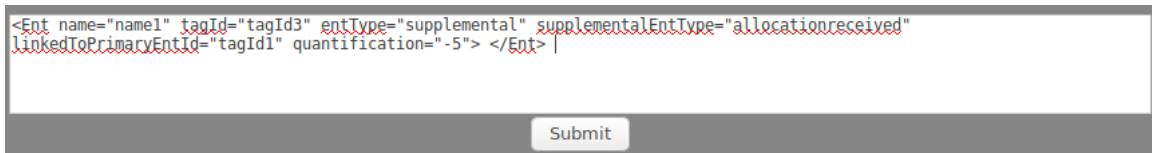


Figure 20. Ent Decrementing Entitlements 2

6) The operation succeeds.

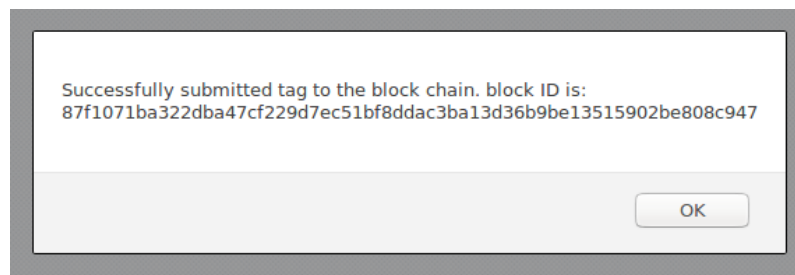


Figure 21. Ent Submission Success 3

7) The user can still submit a request to allocate another -5 assets related to the Initial Ent, even though the balance has reached zero, but since the balance is insufficient, it will fail.

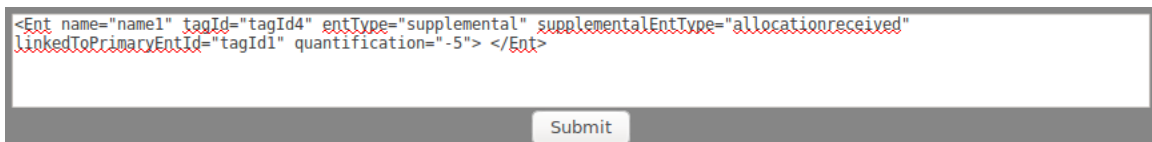


Figure 22. Ent Decrementing Entitlements Below Zero

8) The operation fails.

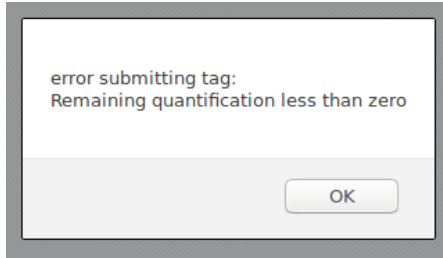


Figure 23. Ent Submission Informative Error

Querying for Ents is structured identically to querying for SWID tags, except different indexes are used. The query below shows that the successful transactions were stored to the ledger and does not show the unsuccessful transaction was not.

A screenshot of the SAM Blockchain web application. The header is "SAM Blockchain" in white on a black background. Below the header is a navigation bar with tabs: "Introduction", "Enter SWID Tag", "Query SWID Tags", "Enter Ent Tag", and "Query Ent Tags". The "Query Ent Tags" tab is active. The main content area has the heading "Query Ent Tags" and contains explanatory text about Ent IDs and linked tags. Below the text is a form with a dropdown menu set to "By Linked Initial Tag", a text input field containing "tagid1", and a "Submit" button. The output of the query is shown as XML:

```
<Ent name="name1" tagid="tagid1" entType="initial" supplementalEntType="" quantification="10" linkedToPrimaryEntId=""></Ent>
<Ent name="name1" tagid="tagid2" entType="supplemental" supplementalEntType="allocationreceived" quantification="-5" linkedToPrimaryEntId="tagid1"></Ent>
<Ent name="name1" tagid="tagid3" entType="supplemental" supplementalEntType="allocationreceived" quantification="-5" linkedToPrimaryEntId="tagid1"></Ent>
```

 At the bottom of the page, there is a footer with "Jared Swanson" on the left, "© Copyright 2017" in the center, and "Last updated 17 December 2017" on the right.

Figure 24. Ent Query Page

Chapter 8: Future Capabilities

The vision of a SWID and Ent blockchain registry is not a single, demonstrable capability. Rather, it is a platform, on top of which a broad range of new tools and new capabilities can emerge. In addition to providing information for IT consumers, it can be used by different vendors to produce a large array of tools and services, and extensible channels can support a broad range of new business models. These sections describe, at a notional level, several select capabilities that could be built on top of the platform.

8.1 Peer-to-Peer Information Sharing

Peer-to-Peer information sharing can occur easily through properly formatted SWID tags. Organizations can share information directly through the chain itself by issuing tags. In many cases, organizations will not trust the content, quality, or accuracy on many or most of the tags on the chain. However, if an organization were to recognize the public key of a known organization, they could, through a tool of their choice, choose to trust the information on the chain signed by the organization's key. Then, for example, if a trusted vendor issues licenses to a third party with rights to sell those licenses, and those licenses pass through multiple organizations, a potential new customer can purchase the licenses, knowing that they are valid because the authenticity of that license can be traced back to the initial transaction where the trusted vendor signed the original transfer of licenses.

8.2 Certificate Authority for Trusted Software

Certificate Authorities could serve to centralize the effort required to distinguish which peers are trustworthy on a blockchain. Using the SWID and Ent registry smart contracts, we have already verified that tags are properly formatted as specified in ISO Standards 19770-2 and 19770-3. There are many other aspects of software tags that users may need to trust, such as quality, completeness, consistency, reputation etc. No single trust authority can be expected to determine what is needed for all users. For instance, technologies that are broadly trusted in China and that are certified by a body like the government may not be trusted by political dissidents. Likewise, software for use in the construction of a control system in a military unmanned aerial vehicle may have a more restrictive set of trusted tag providers than a store clerk trying to figure out which mobile phone applications won't inconvenience her with adware.

In many cases, individually selecting the peers whose tagging practices and whose whose software can be trusted is not feasible. Certificate Authorities already exist that will vouch for the credibility of different organizations. One approach to make trust determinations more scalable would be to integrate tools that use the SWID and Ent blockchain with the existing system of certificate authorities. This could allow the same process, and perhaps even the same keys and infrastructures, to be used to facilitate the selection of trusted peers on the SWID and Ent blockchains.

8.3 Reputation Scoring

Reputation Scoring is a method to determine a level of trust for some entity based on a number of indicators. In the cybersecurity space, it has received substantial attention as a way to select an acceptable level of risk by determining how to apply automated policies to entities, such as websites or executable files, when they don't fall clearly into "known good" or "known bad" categories. Reputation scores can be generated by taking one or more indicators and running them through an algorithm in order to determine some index for trustworthiness of an entity. Based on this index, organizations can educate their business decisions, such as determining which websites their employees will be allowed to access, or which software can be allowed on their networks.

The algorithms that underlie a reputation scoring scheme can be very simple or highly complex. An example on the simple (and sub-optimal) side could be how many users have registered their own tags for a product. On the complex side, multiple data sources could be combined with machine learning techniques to predict the probability of a security breach. In practice, there is rarely enough structured data to effectively use these more advanced techniques. More and better data can lead to more predictive reputation scores. The SWID registry could help provide a publicly accessible store of data with established software attributes, such as common identifiers for software, patches, configurations that could help correlate information for more advanced calculations.

8.4 Software Whitelisting on Endpoints

1. Many users in enterprise systems are granted limited permissions to their machines. One implementation of these policies is an application whitelist, which prevents users from executing programs that have not been explicitly approved (listed on the whitelist). Often these are difficult to update with new patches and updates, and unforeseen dependencies can also make downloading of necessary programs prohibitively difficult. Many organizations will make due with a commercially-provided blacklist, which contains signatures of known malicious software. The most common of these are often referred to as “anti-virus” products. These often use massive databases, and in many cases can be easily circumvented through the use of polymorphic malware. Whitelisting is generally considered a more secure (albeit, more restrictive) approach. A registry of SWID tags and Ents could provide a key platform for whitelisting software that could provide many more customization capabilities, easier configuration, and more automated software whitelisting practices, thereby reducing the disadvantages, and retaining the robust advantages of whitelisting approaches.

Current whitelist technologies generally accept a list of applications that are permitted to run in a certain environment. This basic use case could easily be permitted by a tool that references SWID tags and/or Ents on the blockchain to determine if it is acceptable from both a security and commercial standpoint for certain programs to remain in the environment. While tags may contain more data than is necessary for this

simple use case, it is highly feasible. By using ISO standardized tags, greater interoperability could enable more portability between environments and between tools.

In some cases it may be difficult or undesirable to tag each software component. In these cases we can take advantage of the ability of ISO 19770 family of standards to “bundle” software together. For example, many organizations like to start from a secure baseline operating system image. From that image, they add additional applications as needed. The entirety of an operating system can be included on one tag, which could eliminate the need to tag thousands of operating system components. Then additional tags can be used to describe any programs that are not contained in the base image, still achieving the benefits of a fully tagged infrastructure.

Information sharing could make tags available for both software components and software bundles using the three trust relationships described earlier, and could be used to make the cost of maintaining a full whitelist of software components more feasible. These include peer-to-peer trust relationships, trust authorities, or reputation scoring to automate the process of distinguishing between trusted and untrusted software. Any permutation of these approaches could be used in the same whitelisting scheme and can address both software identification, and software licensing issues.

8.5 Software Discovery

Numerous auto-discovery tools are used to scan networks for IT assets and software, but, as research firms like Gartner acknowledge, only “a limited number... should be used for IT asset management purposes” (Adams, July 2014). This caution stems from the low reliability of the information gleaned from these tools. In other cases,

cybersecurity tools from major vendors can be repurposed to gain limited insight. For example, many security orchestrators can support extensions that can yield slightly more information about each host the software is installed on.

Given these limitations, tools will often use heuristic approaches to determine probabilities that discovered software is a certain application. This approach leaves a major gray area of uncertainty over which applications are running on the network and where they are located. This is much less secure than a system that can definitively identify software packages down to the service pack, the update, even determine the order in which the updates were installed.

According to the ISO 19770-2 standard, software producers, organizations using the software, or third party tag producers can all issue tags. By sharing information over a central repository between trusted tag users, the process of determining the identity of many, or all applications on a network may be possible in a highly automated way. The use of peer-to-peer or trust authority models to acquire trusted tags could allow tags generated at the Enterprise, Industry Sector, Technology, or SecAAS Level to be shared across industry. This creates a situation where a tag can be issued once, and be shared across all tag users, rather than relying on each organization to create and/or manage its own sets of tags.

The ISO Standard 19770-2 defines an “evidence” field to describe unknown programs based on observed characteristics, including the hash of the executable file. This field is intended for use in tags where the identity of the software is unknown. The challenge then becomes a question of correlating existing tags for known software with tags for unknown software. This correlation effort could also leverage the same

information sharing model that leverages whole industries to build out a common repository of information benefit of all. This information could be included on the SWID registry through the issuance of Supplemental Tags.

This approach would likely be highly effective for programs that operate on abstracted platforms. For instance, a hash of a docker container will be identical, since it is running on the docker platform, or a hash of a Java executable of a given version will always be the same because it runs on the standard Java virtual machine. This will also work well for interpreted programs. As long as a script for an interpreted program remains unaltered, it will always have the same hash regardless of the system it runs on. This covers a large number of applications. In each of these cases, a single “evidence” field value could be mapped to a single unique software ID in the SWID registry, thereby also connecting it to any other data resources and tools that can be linked to tags in the registry.

This mapping becomes more difficult in the context of compiled programs that do not have a standard abstraction layer, such as running on a common runtime environment, but there are many possible approaches to deal with these cases. First, while installed applications vary based on factors such as the CPU instruction set of the machine it is loaded onto, the compiler used to compile it, and so on, the content of the installation program is still consistent. According to ISO Standard 19970-2, installation programs get their own type of SWID tag called a Corpus Tag. The hash of installation media can be used for a one-to-one mapping between programs that may be compiled in various ways, and the unique software ID for that program. To ensure this identification isn’t lost, as a program is installed, some tools may be able to map the hash of the unique binary to the

appropriate software ID for later reference to ensure that all programs have appropriate tags.

This does not work, however, when the program has already been installed on a machine and the installation programs are not available, or are not traceable to the installed programs. One way to handle this case is to allow many-to-one mappings between the “evidence” field for unknown tagged software, and software identifiers. The number of relationships to store all permutations for an installed program across all hardware, compilers, compiler settings, tag installations, orders of tag installations, etc. would be prohibitively large. However, if a very large number of the instances of binary files are created in a similar way, for example if the most common programs run on only a few hundred models of CPU, using a small set of different compilers and compiler settings (perhaps defined by the common installation program), and altered by a common ordering of applied patches, then a few hundred or a few thousand values for “evidence” fields may be able to positively correlate them for the vast majority cases where these mappings may be needed. Due to the scaling challenges for this type of application, it would be better to store these data in a mutable data structure. A large number of mappings may be needed, and determining which mappings are most commonly needed likely means that an eviction algorithm, such as Least Frequently Used with Dynamic Aging (LFUDA), would be needed to identify and retain commonly requested variants and evict hashes for binary files that are rarely used.

8.6 Decomposition of Application Dependencies for Digital Rights Assurance

Tracking software dependencies can be very difficult in the software development process. Software products often make use of a large number of existing software components and libraries to reduce the amount of work necessary to develop new services. Often these existing components have terms of use and other licensing restrictions, especially for incorporation in other commercial software. It can be easy to lose track of these components in large software projects. The ability to scan the project and identify all of its components means that any tagged entitlement information on those products can also be recovered and accounted for throughout the process to ensure that the product does not infringe on any licensing requirements.

Chapter 9: Conclusion

Creating an application for a SWID and Ent registry on a blockchain implementation is feasible and practical. Blockchain, however is often used to refer to multiple ideas. First is the blockchain itself that ensures the integrity of data by cryptographically linking each block to the previous block. The second is distribution of the data among multiple participants to ensure redundancy and transparency. The last is the use of consensus algorithms to resolve conflicts when mutually distrustful parties disagree. The SWID and Ent registries do not require a blockchain for implementation, but the cryptographic blockchain, and distributed structures have clear benefits. Since the blockchain can be transparent, a single party can determine the order in which blocks are added, and remove the need for a consensus algorithm.

Hyperledger Fabric seems like a good fit for this kind of system. It uses a blockchain, allows data to be distributed across many peers, and centralizes the function to determine the order for blocks to be added to the blockchain. Unfortunately, Hyperledger Fabric does not seem mature enough to support the registries at this time, and has architectural incompatibilities with this model. Creating a working application for production use would require refactoring, likely replacing the Hyperledger Fabric platform with a solution with more stability and better documentation.

Nonetheless, the results of this proof of concept found that creating blockchain-based registries for SWID tags and Ent is feasible, even though the use of Hyperledger Fabric is problematic for that use case. All of the issues encountered were due to the

Hyperledger Fabric implementation of blockchain, but none of those issues applied to blockchain technologies in general. Therefore, while implementing SWID tag and Ent registries was problematic in this case, the concept of using blockchain to create these registries is still promising.

A simple blockchain, lacking support for distribution or a consensus algorithm can be implemented in all major database platforms. A smaller, but still sizable number of database applications support data synchronization enabling the creation of a distributed platform. The system will still require some kind of service to determine the order records should take. However, if all records in the registry are transparent and users can validate every record with the digital signatures of the data contributors, centralizing the function for sequencing record entries is probably acceptable. If the organization ordering the data is malicious, it could not corrupt data already on the blockchain. It could only prevent new valid records from being added. Therefore, a more mature distributed database with a simple ordering system could be a good option for a better implementation that would require much less work and complexity than the design used here.

The research conducted was not able to investigate the potential of a blockchain registry of SWID tags and Ents to help achieve a high enough level of adoption to make software asset tagging cost effective for software consumers. However, the background research still indicates that there is a lot of potential for this approach. Done well, software asset tags could provide great value to a large number of consumers, and registries could provide an essential foundation to support a diversity new cyber capabilities.

References

- Abdrashitov, Oleg. 2017 November 26-28. Group Training.
- Blockchain.info. (2018 January 14). *Blockchain: Market Capitalization*. Retrieved from <https://blockchain.info/charts/market-cap>
- Cheikes, B. A. Feldman, L. Waltermire, D. Witte, Greg. (2016). *Guidelines for the Creation of Interoperable Software Identification (SWID) Tags* (NIST Interagency Report 8060). Retrieved from <http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf>
- Company Overview of ManageSoft Corporation. *Bloomberg*. Retrieved from <https://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapId=199107>
- DoD Information Technology Standards Registry. (2017). *DISR Standards and Guidance as of January 2017*. Retrieved from <http://www.dsp.dla.mil/Specs-Standards/List-of-DISR-documents>
- Executive Order 13691, 3 C.F.R. (2015).
- Gartner. Inc. (2016). *Gartner Says Organizations Can Cut Software Costs by 30 Percent Using Three Best Practices*. Stamford, CT. Retrieved from <https://www.gartner.com/newsroom/id/3382317>
- Gartner. Inc. (2007). *What You Need to Know About Inventory Tools for IT Asset Management*. Stamford, CT: Adams, P.
- House of Representatives. (2016). H. Rep. No 114-840 § 1653 — *National Defense Authorization Act for Fiscal Year 2017*. Washington, DC: U.S. Government Publishing Office. Retrieved from <https://www.congress.gov/bill/114th-congress/house-bill/4909>
- Hyperledger documentation available at <http://hyperledger-fabric.readthedocs.io>
- International Organization for Standardization. (2012). *Information Technology—Software asset management—Part 1: Processes and tiered assessment of conformance* (ISO/IEC JTC 1/SC 7 Standard No. 19770-1). Retrieved from <https://www.iso.org/standard/52293.html>
- International Organization for Standardization. (2015). *Information technology—Software asset management—Part 2: Software identification tag* (ISO/IEC JTC 1/SC 7 Standard No. 19770-2). Retrieved from <https://www.iso.org/standard/65666.html>

International Organization for Standardization. (2016). *Information Technology—IT asset management—Part 3: Entitlement schema* (ISO/IEC JTC 1/SC 7 Standard No. 19770-3). Retrieved from <https://www.iso.org/standard/52293.html>

Swan, M (2015). *Blockchain: Blueprint for a New Economy*. Sebastopol, CA: O'Reilly Media Inc.

Tagvault.org. (2018, January 9). *Tools Overview: SWID Tag (19770-2 Support)*. Retrieved from <https://tagvault.org/certification/swid-tools-overview>

Glossary

Behavior-Based Tool In the context of information technology, a utility that is programmed with a baseline understanding of normal behaviors for an environment. The tool will identify any activity that diverges from normal.

Blockchain Multiple definitions are in use. In some contexts, it is a simple design pattern where blocks of data are stored with a defined sequence, and each block contains a cryptographic hash of all of the contents of the previous block, ensuring that any modification of the data is detected. In other contexts, it is used to describe several technologies working together, including the blockchain design described above, distribution of data across many stakeholders, and a consensus algorithm.

Certificate Authority An entity that certifies cryptographic public keys.

Consensus Algorithm That is used to ensure that no participant can compromise the system, and ensure data remains consistent across all users.

Defense-in-Depth An approach to security that involves an overlay of many defensive measures that provide overlapping functions.

Distributed Ledger A list of records that is shared between many parties.

Pseudo-Anonymity a trait where actions or things can associated to an identifier, but identity of the entity linked to that identifier can't be known.

Software Asset Management The practices that involve the management of software assets including acquisition, deployment, use, and removal.

Software Entitlement (Ent) A data schema defined by ISO Standard 19770-3 that links instances of software on a system with the entitlements to use that software.

Software Identification (SWID) A data format and set of standard operations defined by ISO Standard 19970-2 that links data on software to instances of data deployed on systems.

Vendor Lock-In A state where a buyer becomes dependent on a particular vendor for their products or services.