Journal of Parallel and Distributed Computing 163 (2022) 62–82

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Blockchain-based automated and robust cyber security management

Songlin He [a], Eric Ficke [b], Mir Mehedi Ahsan Pritom [b], Huashan Chen [b], Qiang Tang [c], Qian Chen [d], Marcus Pendleton [e], Laurent Njilla [f], Shouhuai Xu [g,*]

[a] Department of Computer Science, New Jersey Institute of Technology, USA
[b] Department of Computer Science, University of Texas at San Antonio, USA
[c] School of Computer Science, University of Sydney, Australia
[d] Department of Electrical and Computer Engineering, University of Texas at San Antonio, USA
[e] U.S. Air Force Research Laboratory, USA
[f] U.S. Air Force 90 COS/CYD, USA
[g] Department of Computer Science, University of Colorado Colorado Springs, USA

ARTICLE INFO

Article history:
Received 2 March 2021
Received in revised form 11 November 2021
Accepted 8 January 2022
Available online 4 February 2022

Keywords:
Cyber security management
Blockchain
Hyperledger fabric
IPFS

ABSTRACT

We initiate the study on the problem of *automated* and *robust* Cyber Security Management (CSM). We exemplify the problem by investigating how CSM should respond to the discovery of cyber intelligence that identifies new *attackers*, *victims*, or *defense capabilities*. Given the complexity of CSM, we divide it into three classes, referred to as Network-centric (N-CSM), Tools-centric (T-CSM) and Application-centric (A-CSM). These lead to a range of functions for examining whether, and to what extent, a network has been compromised. Moreover, we propose to incorporate blockchain (via Hyperledger Fabric) to build a decentralized CSM system, dubbed B2CSM, that ensures the retrieval of valid invocation results for CSM purposes. We also integrate B2CSM with a decentralized storage network (DSN), instantiated by InterPlanetary File System (IPFS), to reduce on-chain storage costs without hindering its robustness. We present the design and implementation of the prototype B2CSM system. Experiments with real-world datasets show that the CSM solutions and system are effective and efficient.

1. Introduction

The importance of enterprise-level cyber security management cannot be overstated. For example, say there is a network administrator named Bob who defends a certain enterprise network. When Bob becomes aware of a new Advanced Persistent Threat (APT) attack that has been active in the wild for a while, he needs to investigate whether or not his network has been a victim of the APT and if so, what the damages are. Indeed, the standard defined in ISO/IEC 27035 includes a five-phase incident management process: prepare, identify, assess, respond and learn [26]. In order to be effective, such standardization must be supported by tools [5]. However, existing tools mainly focus on vulnerability management, incident management, or security information and event management [19,27]. Despite these tools, many routine cyber defense activities are still a *manual* process [5], meaning that defenders cannot respond to cyber events rapidly. Moreover, the manual process is often conducted in isolation because enterprises rarely share cyber intelligence with each other. This problem persists despite the extensive body of work highlighting the importance of sharing cyber threat intelligence [3,5,11,13,41,42]. For example, learning the attacks that have successfully penetrated into some enterprise "A" would undoubtedly help another enterprise "B" defend its network against the same or similar attacks. This observation demonstrates that the problem of effective cyber security management (CSM) remains largely open.

One main challenge encountered when designing an effective CSM system is that of ensuring its *robustness*. For example, a centralized CSM consists of a single point of failure, which often outweighs its performance advantage, especially because CSM itself is clearly an important target of the attackers. This and other possible vulnerabilities naturally motivate the use of distributed or decentralized CSM. However, leveraging a classical distributed database that tolerates crashes for CSM purposes is still problematic because this technique requires one to trust all participants (i.e., *crash fault tolerant* (CFT) [2,44]) but is not resistant to attacks (i.e., Byzantine faults). This highlights the importance of incorporating Byzantine Fault-Tolerance (BFT) into a decentralized CSM.

* Corresponding author.
E-mail address: sxu@uccs.edu (S. Xu).

https://doi.org/10.1016/j.jpdc.2022.01.002
0743-7315/

Another priority is the automation of CSM itself because traditionally CSM-related tasks have been done manually [5], which incurs delays and is tedious and error-prone. In addition, a CSM system should offer other properties such as *accountability*, meaning that the actions of both providers and consumers of cyber intelligence should be observable by each other in order to maintain transparency of intent.

In this paper, we make a significant step towards formalizing CSM by defining three kinds of CSM functions in relation to cyber intelligence sharing, whereby the participating defenders share and leverage cyber intelligence for their CSM purposes. Our contributions can be summarized as the follows.

1.1. Our contributions

We make three contributions. First, we initiate the study of *robust* and *automated* CSM in relation to three types of cyber intelligence: (i) newly detected cyber attackers, which may be leveraged to detect previously unidentified victims; (ii) newly detected victims, which may be leveraged to detect previously unidentified attackers; and (iii) new defense capabilities, which may be leveraged to detect previously unidentified attacks. We further classify CSM functions based on the layer from which their useful intelligence comes: Network-centric CSM (N-CSM), which leverages network-related data for CSM purposes; Tools-centric CSM (T-CSM), which leverages data collected from cyber defense tools for CSM purposes; and Application-centric CSM (A-CSM), which leverages application-specific data for CSM purposes. In order to organize, store and process these cyber data, we propose the abstraction of Annotated Graph Time Series Representation (AGTSR) and introduce algorithms for realizing these CSM functions.

Second, we propose Blockchain-Based CSM (B2CSM) to achieve automated and robust CSM. The design of B2CSM raises a number of challenges. Here three such challenges are highlighted as follows:

- *How to make the proper design choices for the B2CSM system instantiation?* The construction of the B2CSM system involves multiple components such as blockchain type, consensus mechanism, state database etc. We provide concrete analysis and justify our design choices. For example, as there are three classes of CSM functions, there is a spectrum of design options (e.g., one chain for all classes of CSM functions vs. one chain per class of CSM functions) we can use for setting parameters. We explore the advantages and disadvantages of each option in terms of complexity, maintenance workload, and flexibility, and decide to use one chain per class of CSM function.
- *How to deal with a large volume of cyber data?* The CSM data (e.g., network traffic) is often stored in large volumes. However, storing such large data on blockchain directly in the form of transactions and deploying CSM functions as smart contracts would force users or other smart contracts to parse many blocks to extract individual transactions and find the relevant data. This is prohibitively inefficient. We solve this problem by proposing a fine-grained ledger structure in the underlying blockchain platform (i.e., Hyperledger Fabric [6]) to efficiently retrieve large-volume related cyber data, which is split into chunks and stored in Fabric state database. Note that such a design essentially takes advantage of the best features of both blockchain and database (i.e., security vs. performance) [44] by letting a blockchain act as a security layer for the nodes, so that they can reach consensus on the cyber data before it is stored in the state database and can also efficiently and securely retrieve cyber data from the ledger (i.e., the state database) via smart contract.
- *How to reduce ledger storage costs robustly*? Storing cyber data in the state database results in a whole copy of cyber data on each full node (i.e., the server that participates in the consensus protocol) in the blockchain network. To further reduce such a storage cost, we incorporate a Decentralized Storage Network (DSN) instantiated by InterPlanetary File System (IPFS) [7] so that the storage of a huge volume of cyber data is delegated to DSN while each full node only keeps a short reference (i.e., content identifier returned by IPFS) in state database. However, there are still some technical challenges to solve in such a hybrid architecture. For example, how can we cope with loss or tampering of the cyber data before it is submitted to the network? How can we ensure that the output of CSM invocations cannot be manipulated by malicious parties? To handle these problems, we propose leveraging the techniques proposed in [22] to ensure the integrity of the cyber data. Moreover, we ensure the correctness of CSM invocation results by querying from multiple full nodes and validating the attached signatures (as elaborated in Section 3.2.2).

Third, we present a full-stack prototype implementation of B2CSM and measure its performance. All the CSM functions are implemented as smart contracts (i.e., “chaincode”, in the terminology of Fabric). The entire suite of fully-distributed network construction scripts and full-stack implementation code are available online: `https://github.com/Blockchain-World/B2CSM.git`. In order to evaluate the efficiency of the prototype system, we run it on distributed heterogeneous servers with some real-world cybersecurity data relevant to the three classes of CSM functions. Experimental results show that B2CSM is practical. For example, the average query latency conducting a specific N-CSM function ranges from 81.09 ms to 94.23 ms with four nodes. We also experiment with seven and ten nodes, which demonstrate a similar efficiency.

1.2. Related work

CSM related prior studies. Our CSM functions take specified types of threat intelligence as input. Prior studies in threat intelligence sharing can be divided into four categories: (i) characterizing the opportunities and challenges [24,41]; (ii) understanding the legal and regulatory matters [4,45]; (iii) exploring standardization and principles [13,17]; and (iv) developing tools [11,13,40]. Our paper is closely related to the preceding category (iv), but our study is unique because we formulate the problem of *robust* and *automated* CSM and present a *blockchain-based* design and implementation. It is worth mentioning that CSM is different from cyber forensics [33]. This is because forensics is oriented toward certain details, such as attack attribution and criminal investigation, which are not critical in all cases of cyber attacks, and can inhibit the efficient response to active attacks. In contrast, CSM prioritizes efficiency, which is closer to the aim of incident response.

Other blockchain-based decentralized applications. To the best of our knowledge, we are the first to investigate the application of blockchain for the CSM field. This is true despite the existing varieties in which blockchain technology has been leveraged, which include far more applications than just cryptocurrency and smart contracts [35,51]. Some of these include: enhancing integrity and privacy of shared cyber security data [39]; replacing conventional trusted third parties to ensure fairness for peer-to-peer content delivery networks [23]; managing public key certificate and revocation status [37]; enhancing the trustworthiness of cryptographic digital signatures in the presence of compromised private signing keys [52,53]; facilitating data integrity and data management in IoT systems [22]; detecting violations of access control policies in cloud environments [16]; managing data provenance, accountabil-

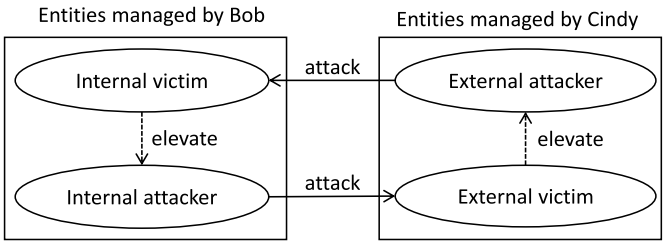


Fig. 1. Illustration of external vs. internal attacker and victim from defender Bob's (rather than defender Cindy's) point of view.

ity and copyright protection [29]; enabling data sharing [39] and decentralized crowdsourcing services [32].

1.2.1. Paper outline

Section 2 describes CSM model, functions and data structures. Section 3 presents the design of B2CSM and the security analysis. Moreover, we introduce the implementation of a B2CSM prototype and evaluate its performance based on real cyber data. Section 4 discusses the limitations of the present study, and Section 5 concludes the present paper.

2. CSM model, functions and data structures

Terminology. A cyber defender, Bob, manages a set of entities, which are broadly defined to accommodate computers and other objects of cybersecurity significance. As illustrated in Fig. 1, we make the distinction between *external* entities (i.e., those not managed by Bob but which may be managed by another defender, Cindy) and *internal* entities (i.e., those managed by Bob); this external vs. internal distinction is from a specific defender's point of view, in this case, Bob's. An entity can be in one of three states: *victim*, *attacker*, or *normal*. A *victim* entity is one that has been compromised by an external or internal *attacker* entity; an *attacker* entity is one that exhibits malicious behavior; and a *normal* entity is one that is neither a victim nor an attacker entity. A normal entity can become a victim entity when it is attacked by an external or internal attacker entity, and a victim entity can elevate to an attacker entity.

2.1. CSM model

In the CSM model, a defender Bob, or more precisely his CSM App (CSMA), leverages some input cyber intelligence to identify victim and attacker entities, where the input intelligence may be (i) shared by other defenders or (ii) discovered by some cyber defense tools used by the defender Bob. In what follows, we describe five kinds of cyber intelligence, three classes of CSM functions, and a general data structure designed to facilitate these CSM functions.

2.1.1. Input cyber intelligence

As illustrated in Fig. 2, we consider five kinds of input cyber intelligence, which are prefixed by 'I-'.

(I-1A) Intelligence that points to some *external attackers*, possibly accompanied by the time window during which an external attacker is active.

(I-1B) Intelligence that points to some *internal attackers*, which may have attacked some external victims and been detected by another defender, or some internal victims and been detected by some cyber defense tools used by Bob.

(I-2A) Intelligence that points to some *external victims*, which have been attacked by some internal or external attackers.

(I-2B) Intelligence that points to some *internal victims*, which have been attacked by some internal or external attackers. The intelligence may be collected, for example, by the leakage of data specific to the victim (e.g., social security numbers or passwords) or by a cyber defense tool (e.g., intrusion detection system or anti-malware tool).

(I-3) Intelligence that points to some *new* defense capabilities, such as methods for detecting previously undetected attacks (e.g., 0-day attacks).

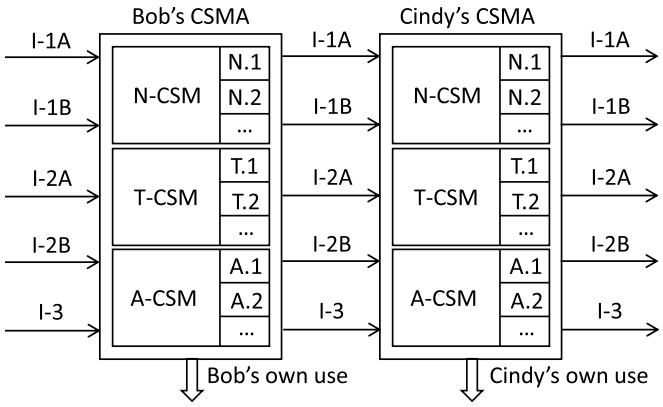


Fig. 2. CSM model with five kinds of input cyber intelligence and three classes of CSM functions. The naming scheme is: '1' means attacker, '2' means victim, and '3' means capabilities; 'A' means intelligence on external entities and 'B' means intelligence on internal entities.

2.1.2. An overview of three classes of CSM functions

As depicted in Fig. 2, Bob's CSMA takes as input some cyber intelligence and the relevant cyber data, uses the CSM functions (specified below) to identify other internal or external attackers/victims, and outputs the resulting intelligence. Bob may choose to share this output with another defender, say Cindy, about his internal or external attackers/victims (i.e., input cyber intelligence I-1A, I-1B, I-2A, I-2B from Cindy's point of view). To be specific, we define three classes of CSM functions, as shown in Fig. 2: (i) Network-centric CSM (N-CSM), which leverages network-related data and cyber intelligence for CSM purposes; (ii) Tools-centric CSM (T-CSM), which leverages data collected from cyber defense tools and cyber intelligence for CSM purposes; and (iii) Application-centric CSM (A-CSM), which leverages application-specific data and cyber intelligence for CSM purposes. Each class contains multiple CSM functions, and the core ideas of these functions are described below.

N-CSM. N-CSM functions are centered at examining the input cyber intelligence against network traffic data, which may be collected at a gateway between the external network (e.g., the Internet) and the internal network (e.g., an enterprise network). Network traffic data can be represented by IP packets and TCP/UDP flows, which incur different costs on storage. We define the following three N-CSM functions.

(N.1) This function is designed to identify internal victims of some external attackers, which are given as the input cyber intelligence (i.e., input I-1A). Specifically, at time t', Bob is given cyber intelligence that an *external attacker*, identifiable by its IP address, attacker_IP, was active at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Bob needs to identify his internal systems that may have been compromised by the external attacker in time interval $[t_1, t_2]$.

(N.2) This function is designed to identify external attackers that may have caused the compromise of some internal victims (i.e., input I-2B). Specifically, at time t', Bob is given cyber intelligence that an *internal victim*, identifiable by victim_IP, was attacked at some point in time interval $[t_1, t_2]$ where

$t' \geq t_2$. Bob needs to identify the external IP addresses that contacted victim_IP in time interval $[t_1, t_2]$.

(N.3) This function is designed to identify potential secondary victims that may have been attacked before, during or after the known compromise of some other *internal victim* (i.e., input I-2B and/or I-1B). Specifically, at time t', Bob is given cyber intelligence that an internal victim IP address, identifiable by its IP address, victim_IP, was attacked at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify other victims that were contacted by the potential attackers that may have compromised the given victim_IP during time interval $[t_1, t_2]$.

T-CSM. T-CSM functions are centered at cyber defense tools, such as Network-based Intrusion Detection Systems (NIDSs), Host-based Intrusion Detection Systems (HIDSs), and anti-malware systems. These tools often output alerts as indicators of malicious or suspicious activities. We define the following three T-CSM functions.

(T.1) This function is designed to identify the attack path(s) through which a known *internal victim* was compromised (i.e., input I-2B). Specifically, at time t', Bob is given cyber intelligence that an internal victim, say victim_IP, was compromised at some point during the time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the attack path(s) that may have been leveraged to compromise victim_IP.

(T.2) This function identifies victims of zero-day attacks by leveraging a new defense capability (i.e., input I-3). Specifically, at time t', Bob is given cyber intelligence on a new detection method (e.g., signature) for detecting a previously unknown zero-day attack. Then, Bob needs to identify the internal victims that were attacked according to the new detection method during a past time interval $[t_1, t_2]$, $t' \geq t_2$.

(T.3) This function is designed to identify the cascading damage caused by a given attacker (i.e., input I-1A or I-1B). Specifically, at time t', Bob is given cyber intelligence that a malicious external or internal entity was active at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the entities that were directly or recursively accessed by the malicious entity during time interval $[t_1, t_2]$.

A-CSM. A-CSM functions are centered at specific applications that are often exploited to wage attacks, such as drive-by downloads via web browsers and spear-phishing via email. As examples, we consider the following three A-CSM functions.

(A.1) This function is designed to identify secondary *internal victims* (e.g., browsers or email clients) that have been targeted by the same attack that succeeded against a known compromised entity (i.e., input I-2B). Specifically, at time t', Bob is given cyber intelligence that an internal entity (i.e., browser or email user) was compromised at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify other internal victim entities (i.e., browsers or email users) that communicated with any of the external attacker (i.e., URLs or email users) that compromised the internal victim during time interval $[t_1, t_2]$.

(A.2) This function is designed to identify *internal victims* (e.g., browsers or email users) of an external attacker (namely input I-1A). Specifically, at time t', Bob is given an external attacker (i.e., URL or email address) that was active at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the other internal victims (i.e., browsers or email users) that may be compromised because they communicated with the external attacker during time interval $[t_1, t_2]$.

(A.3) This function is designed to identify *internal victims* that may be impacted by known attacks against an external victim (e.g., spoofed URL or email address, namely input I-2A). Specifically, at time t', Bob is given cyber intelligence that an external victim (i.e., URL or email address) was spoofed to wage attacks at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the external attackers (i.e., URLs or email addresses) that spoofed the given external victim during time interval $[t_1, t_2]$ and the internal victims (i.e., browsers or email addresses) that communicated with the external attacker during time interval $[t_1, t_2]$.

2.1.3. A general CSM data structure

To realize the CSM functions, proper data representations are needed. We propose a general data structure, known as an *Annotated Graph Time Series Representation* (AGTSR), by dividing the time horizon into $T+1$ *time windows* at some resolution (e.g., hour or day). To reduce the number of notations, we make the following convention: the default use of t, t_1, t_2 refers to specific points in time; we also use the term *time window* t, t_1, t_2 to refer to the t-th, t_1-th, and t_2-th time window, where $0 \leq t, t_1, t_2 \leq T$.

For time window t, we use $G(t) = (V(t), E(t), A(t))$ to represent the relevant cyber activities for CSM purposes, where $V(t)$ is the vertex set with each vertex representing an entity (e.g., IP address, computer or device), $E(t)$ is the arc set with each arc representing some communication activity, and $A(t)$ is the annotation set such that $A(t) = \{A_{uv}(t) : (u, v) \in V(t) \times V(t)\}$ with $A_{uv}(t)$ being a set of annotations associated to $(u, v) \in V(t) \times V(t)$ and $A_{uv}(t)$.count denotes the number of IP packets or TCP/UDP flows along an arc (u, v) in time window t. That is, $A_{uv}(t)$.count $= 0$ means $(u, v) \notin E(t)$ and $A_{uv}(t)$.count > 0 means $(u, v) \in E(t)$, and count is the number of IP packets or TCP/UDP flows from entity (e.g., IP address) u to entity v in time window t. The meanings of annotations in $A_{uv}(t)$ are specific to the class of CSM functions, and will be elaborated below. In principle, $G(t)$ may be stored as an adjacency matrix or list; for simplicity, we will focus on the adjacency matrix representation and $A_{uv}(t)$ can be seen as an extension of the standard adjacency matrix. Our model can support division of a network into subnets with both intra- and inter-subnet communications. We can achieve this by extending $G(t) = (V(t), E(t), A(t))$ of time window t to $G^m(t) = (V^m(t), E^m(t), A^m(t))$, where $V^m(t) \subseteq V(t)$ are the nodes belong to a subnet and formulate a partition of $V(t)$, $(u, v) \in E^m(t)$ means $u, v \in V^m(t)$, and A^m_{uv} means $u, v \in V^m(t)$. There are also arcs $E^{m,m'}(t) = \{(u, v) : u \in V^m(t), v \in V^{m'}(t)\}$. The cybersecurity meanings of these notations are specific to the CSM functions in question and thus elaborated later.

Besides, we define $[n] = \{1, \ldots, n\}$, and use $\max_{t \in [t_1, t_2]} |V(t)|$ to denote the maximum number of entities (e.g., computers, IP addresses, or browsers) during a time window in between time window t_1 and time window t_2, namely $\max_{t \in [t_1, t_2]} |V(t)| = \max\{|V(t_1)|, |V(t_1 + 1)|, \ldots, |V(t_2)|\}$ with $0 \leq t_1 \leq t_2 \leq T$. Similarly, we define that $\max_{t \in [t_1, t_2]} |V^m(t)| = \max\{|V^m(t_1)|, |V^m(t_1 + 1)|, \ldots, |V^m(t_2)|\}$.

2.2. CSM data structures and functions

We now present the three concrete CSM data structures and functions, and the detailed algorithms are deferred to Appendix A. Note that all these algorithms are implemented in the smart contract (i.e., chaincode) to facilitate CSM.

2.2.1. N-CSM data structure and functions

For N-CSM, AGTSR can accommodate network communications such that a node $u \in V(t)$ represents a computer, and an arc $(u, v) \in E(t)$ represents the communications between nodes u and

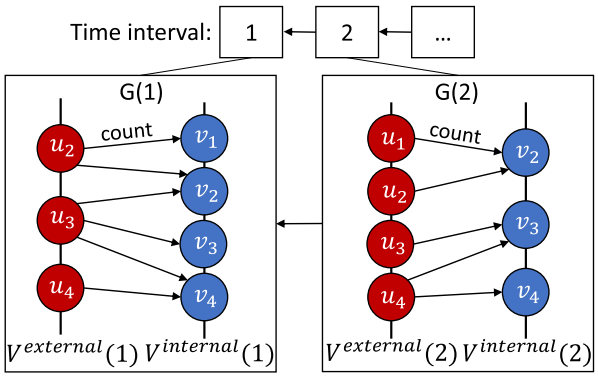


Fig. 3. Data structure for N-CSM.

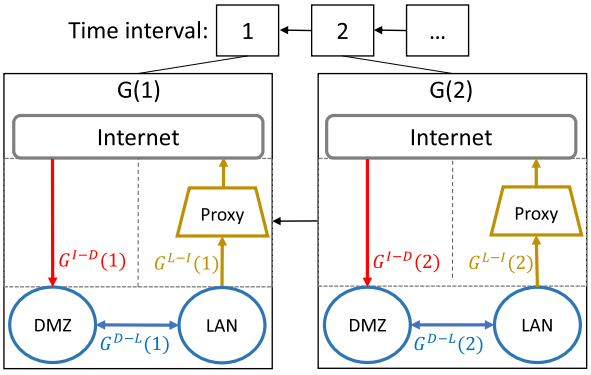


Fig. 4. Data structure for T-CSM.

v initiated by u. In N-CSM, we are often concerned with border communications, meaning the communications between the internal entities and the external entities. In this case, $V(t)$ is partitioned into V^{external} and V^{internal}, where V^{external} is the set of external entities (e.g., IP addresses) and V^{internal} is the set of internal entities. For time window t, there is a $G(t) = (V(t), E(t), A(t))$ as defined above. Fig. 3 illustrates $G(1), G(2), \ldots$; for example, we have $u_2, u_3, u_4 \in V^{\text{external}}(1)$ and $v_1, v_2, v_3, v_4 \in V^{\text{internal}}(1)$ where count is only illustrated for $(u_2, v_1) \in E(1)$ for a better visual effect.

The N-CSM algorithm details are deferred to Appendix A.1 where Algorithm 1 realizes N-CSM function N.1 by identifying victims given an attacker; Algorithm 2 realizes N-CSM function N.2 by identifying potential attackers based on their communications to a given victim; Algorithm 3 realizes N-CSM function N.3 by identifying secondary victims of the attacker that compromised the input victim.

2.2.2. T-CSM data structure and functions

For T-CSM, Fig. 4 shows an example network to clearly convey the ideas. The network has three disjoint subnets: the Internet (i.e., the external subnet), the demilitarized zone for external-facing servers (DMZ), and the local area network (LAN). This suggests that Bob can use (i) an AGTSR to represent the interactions between the Internet and the DMZ, or $G^{\text{I-D}}(t)$ for short; (ii) an AGTSR to represent the interactions between the LAN and the Internet, or $G^{\text{L-I}}(t)$ for short; and (iii) an AGTSR to represent the interactions within the DMZ itself, within the LAN itself and between the border of the DMZ and the LAN, or $G^{\text{D-L}}(t)$ for short. Note that $V(t) = V^{\text{I-D}}(t) \cup V^{\text{L-I}}(t) \cup V^{\text{D-L}}(t)$. In T-CSM, the annotation of an arc is a list of alerts (i.e., $A_{uv}(t) = \{\text{alerts}\}$). These alerts are triggered by the traffic across each arc, which often corresponds to a routing path rather than a physical link.

The T-CSM algorithm details are deferred to Appendix A.2 where Algorithm 4 realizes T-CSM function T.1 by inferring the attack paths to the compromised internal entity; Algorithm 5 realizes T-CSM function T.2 by retrospectively detecting the victims of a zero-day attack during a past time window prior to discovery of the zero-day attack; Algorithm 6 realizes T-CSM function T.3 by identifying the cascading damage of a given attack, specifically, it determines which entities were targeted by the given attacker, either directly or recursively.

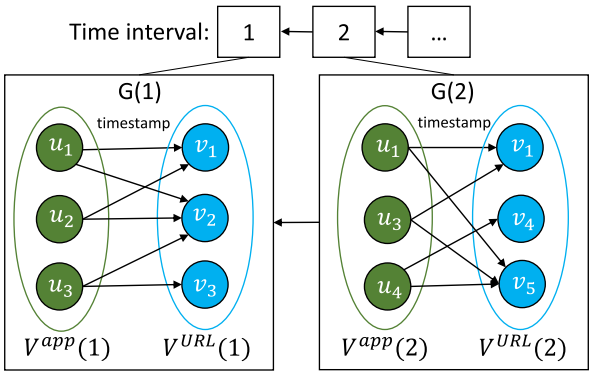


Fig. 5. Data structure for A-CSM.

2.2.3. A-CSM data structure and functions

For A-CSM, we use the example of web applications, but the discussion can be adapted to accommodate other applications, e.g., email systems. In this example, browsers (or their IP addresses) are internal entities and URLs are external entities. As illustrated in Fig. 5, $G(t) = (V(t), E(t), A(t))$, where $V(t) = V^{\text{app}}(t) \cup V^{\text{URL}}(t)$, $E(t)$ is the arc set such that arc $(u, v) \in E(t)$ means browser $u \in V^{\text{app}}(t)$ visited URL $v \in V^{\text{URL}}(t)$ in time window t, each arc $(u, v) \in E(t)$ is annotated with a timestamp $\in A_{uv}(t)$, where a value of -1 means $(u, v) \notin E(t)$.

The A-CSM algorithm details are deferred to Appendix A.3 where Algorithm 7 realizes A-CSM function A.1 by identifying suspicious internal applications (i.e., potentially compromised browsers); Algorithm 8 realizes A-CSM function A.2 by identifying victim browsers given a known malicious URL; Algorithm 9 realizes A-CSM function A.3 by identifying victim browsers of spoofed (e.g., typo-squatted) URLs given the input of an abused URL with url_id.

3. B2CSM system and evaluation

A straightforward realization of the CSM model depicted in Fig. 2 would let each defender build a centralized cyber security management system (mainly for the sake of efficiency) to maintain their own cyber data and perform CSM invocation due to some received threat intelligence. However, such a design is insufficient due to the considerations that: (i) centralized architectures typically pose the risk of single-point-of-failure, and therefore a decentralized system would be preferable to tolerate crash faults; (ii) when considering a decentralized system, even in the network of the same enterprise that the defender manages, it is still possible that some of the servers get corrupted, hence not only crash fault tolerance (CFT) but also byzantine fault tolerance (BFT) are needed to construct a robust CSM system; (iii) besides the robust storage of cyber data, the invocation records (e.g., which party invoked a CSM function at a certain point in time) are also potentially valuable to realize accountability, and these records should be tamper-proof against malicious actions; (vi) the decentralized system is expected to correctly execute some pre-defined operations, i.e., the CSM functions, in an automated manner instead of the involvement of manual procedure. To overcome these challenges, we propose leveraging blockchain to build a decentralized,

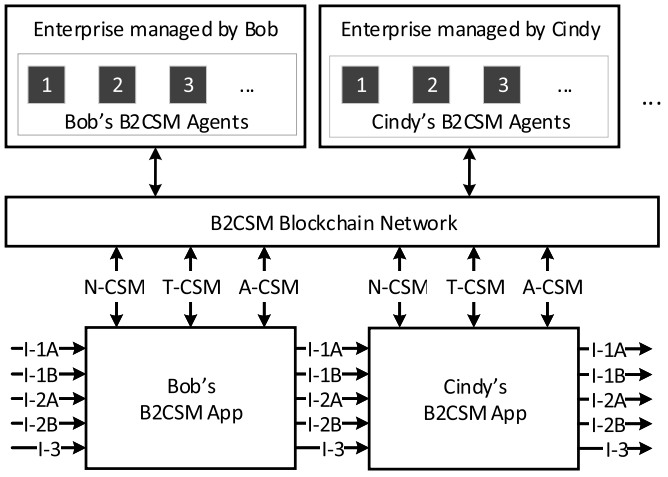


Fig. 6. The B2CSM model extending from the CSM model.

automated and robust blockchain-based CSM system, leading to B2CSM. In what follows, we present the key designs of B2CSM; instantiate it atop the blockchain platform; analyze its security properties and evaluate its performance based on real cyber data.

3.1. B2CSM model and architecture

Fig. 6 highlights the B2CSM model extended from the CSM model by storing cyber data, $G(t)$'s, in the B2CSM blockchain network and incorporating B2CSM Apps and B2CSM Agents. B2CSM Apps are the interface for defenders to run CSM functions, by (i) taking as input some cyber intelligence and identifiers (e.g., a time frame) of the relevant cyber data and (ii) presenting the output of the CSM functions to the defender. B2CSM Agents collect cyber data and write the data to the B2CSM blockchain network.

The B2CSM model described in Fig. 6 lends itself to the B2CSM architecture depicted in Fig. 7, which is presented from the defenders' perspective. In this architecture, a defender uses a set of B2CSM agents to collect cyber data from the enterprise network. These agents write the collected data into the defender's B2CSM blockchain network. The defender interacts with their B2CSM App to execute CSM functions with some input cyber intelligence. The CSM functions run in the form of smart contract at full nodes in the B2CSM blockchain network. The B2CSM middleware acts as an intermediary between the B2CSM App and the blockchain network. To provide permissioned access control, the defenders are identified via a Certificate Authority (CA). These components interact with each other to form the B2CSM system.

3.2. B2CSM system design and security analysis

3.2.1. Instantiating the architecture as a system

The B2CSM architecture in Fig. 7 can be instantiated into B2CSM systems in different ways. We now propose a concrete instance by providing needed design choices:

Decision on blockchain type. We propose using the *permissioned* blockchain [6] to realize B2CSM due to the following reasons: (i) CSM needs to authenticate its participants and users because cyber security management copes with sensitive data; (ii) only parties who are interested in CSM (e.g., honest defenders share a common goal of protecting their systems from malicious attacks) need to participate in, and therefore it is unnecessary to be public to all; (iii) the participating entities may not fully trust each other, highlighting the importance of achieving accountability.

Note that permissioned blockchains can be further divided into *private* blockchains, where the full nodes in the blockchain network belong to one enterprise, and *consortium* blockchains, where the full nodes are managed by multiple enterprises.

Decision on the number of chains and their types. We propose using one chain per class of the three classes of CSM functions. The Fabric *channel* mechanism offers this service and creates a separated "subnet" containing its joined members, its ordering service nodes, a shared ledger, and the application chaincodes. One-chain-per-class provides a modular structure for the B2CSM network and allows for flexible extension of more potential CSM functions. In B2CSM, we propose two kinds of chains (or channels):

- **Private chain/channel for storage**: A defender of each enterprise can create a *private* channel to store its own cyber data and perform different CSM functions, leading to a (permissioned) *private* B2CSM blockchain.
- **Consortium chain/channel for sharing**: Defenders can also jointly create a channel for storing cyber data (as secret shares [43] or encrypted) and sharing their cyber intelligence data, leading to a (permissioned) *consortium* B2CSM blockchain.

In both cases, each channel maintains a unique *ledger*, which consists of a *blockchain* for *on-chain* data storage (as transactions) and *state database* for *off-chain* data storage (as key-value pairs), and can serve a specific CSM class, namely N-CSM, T-CSM or A-CSM.

Fig. 8 depicts the channel architecture of B2CSM. The defender of an enterprise can create a private channel by only allowing the server nodes managed by the defender to join in. Intuitively, the cyber data of an enterprise is maintained and accessed only by its own servers, which jointly maintain a distributed ledger. Moreover, defenders of different enterprises can create a consortium channel for sharing their cyber intelligence data such as the outputs of CSM function invocations. Following a general BFT consensus security model [12], the number of full nodes $\mathcal{N}$ in any channel satisfies $\mathcal{N} \geq 3f + 1$, where f is the number of faulty nodes that can be tolerated.

Decision on the consensus protocol to use. Since our threat model considers compromised blockchain network nodes, we need to make B2CSM achieve Byzantine Fault Tolerant (BFT). The Ordering Service Nodes (OSNs) in Fabric are external nodes (i.e., rather than the blockchain's full nodes) and that the ordering service only supports Crash Fault-Tolerance (CFT) consensus mechanisms such as Zookeeper with Kafka or Raft [25]. To achieve BFT, we propose integrating the work in [46], known as BFT-SMaRt. Moreover, we propose running the ordering service at the full nodes of the B2CSM blockchain, instead of delegating this service to extra nodes.

Decision on the state database to use. Fabric supports *leveldb* and *couchdb* as state databases. Although both support key-value storage, couchdb offers rich queries (e.g., the value can be JSON format whereas leveldb only supports string-based queries). In light of this, we adopt couchdb as the B2CSM state database and the concrete data format is elaborated in Section 3.2.2.

Decision on the locality of the B2CSM middleware. We propose running the middleware at every B2CSM blockchain full node. The middleware has multiple sub-functions, such as formatting a defender's invocation of CSM functions, interacting with the B2CSM blockchain network, and polishing the output of CSM functions before returning it to the B2CSM App. These services are important because (i) different kinds of CSM functions may require different kinds of data pre-processing, and (ii) the middleware serves as an intermediate level of abstraction to support extensive functionalities that may emerge in the future.

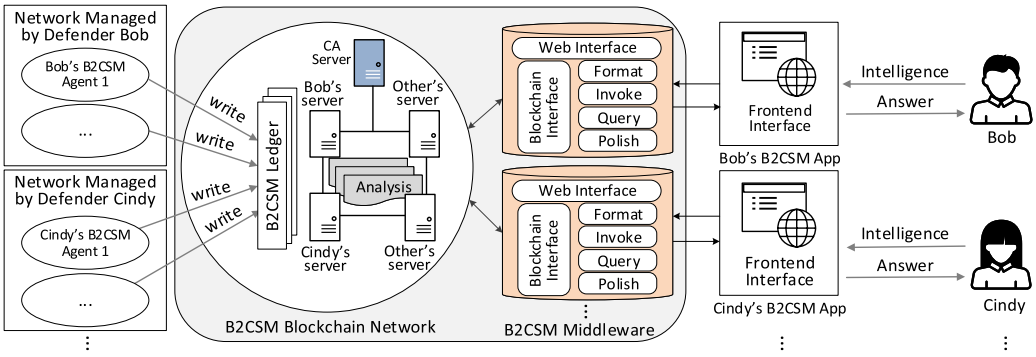


Fig. 7. Illustration of B2CSM architecture. Each defender has a set of agents for collecting cyber data and writing into the B2CSM blockchain (either via the B2CSM Middleware or an independent writing module). Each defender runs a B2CSM App, which invokes CSM functions deployed in the smart contract using cyber intelligence from the defender.

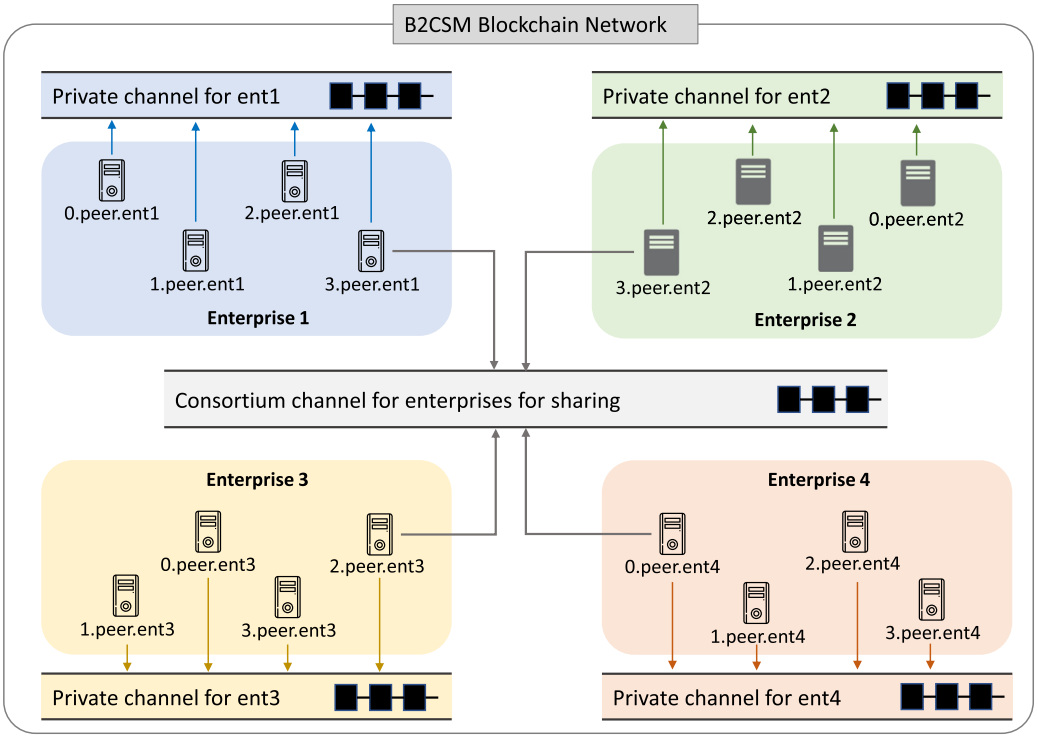


Fig. 8. The channel architecture in B2CSM blockchain network. The private and consortium channels can further serve different classes of CSM, i.e., N-CSM, T-CSM or A-CSM.

3.2.2. B2CSM system design

In light of the design choices above, we now present the B2CSM system design, which contains two main phases: *cyber data replication* and *CSM function invocation*. We elaborate the key challenges in each phase and the corresponding solutions at a high-level. The concrete message flows are deferred to the specified appendices.

Phase I: Cyber data replication. This phase allows a defender, Bob, to robustly store the cyber data via private channels in the B2CSM blockchain network. Upon system setup, defenders can continuously write collected cyber data to the channel via B2CSM agents, i.e., the local servers managed by Bob. The key challenge lies in *dealing with a large volume of cyber data* in terms of efficient writing, reading and robust storage. We propose two methods to solve this challenge and analyze their advantages and disadvantages. Specifically:

Method 1 ($\mathcal{M}_1$): Splitting into chunks with fine-grained ledger structure. The handling of a large volume of cyber data $G(t)$ requires efficient *uploading* and *retrieval*. First, to replicate a large volume of cyber data to the blockchain network, we propose dividing $G(t)$ into small data units and uploading them in parallel. Second, the efficiency of data retrieval largely depends on how the data units are stored in the B2CSM network. While it is tempting to store all of the cyber data on the blockchain in the form of transactions and use smart contracts to point to which blocks contain the relevant data units for what time window, this design will incur large latency when multiple blocks need to be traversed. Besides, a block may contain data units belonging to different time windows as the block size is fixed when initializing a channel, leading to possible retrieval of irrelevant cyber data. This prompts us to propose a proper ledger structure by extending the Fabric state database: the blockchain full nodes not only reach consensus on data units and package them into consecutive blocks, but also proactively update the state database for later efficient retrieval purposes.

Fig. 9 depicts the structure of the B2CSM ledger, where the *blockchain* stores two kinds of transactions: (i) the transactions containing the history of cyber data replication (i.e., who submits which cyber data to the B2CSM blockchain network); (ii) the transactions containing the history of CSM function invocations for auditing purposes. The *state database* stores real cyber data $G(t)$ as

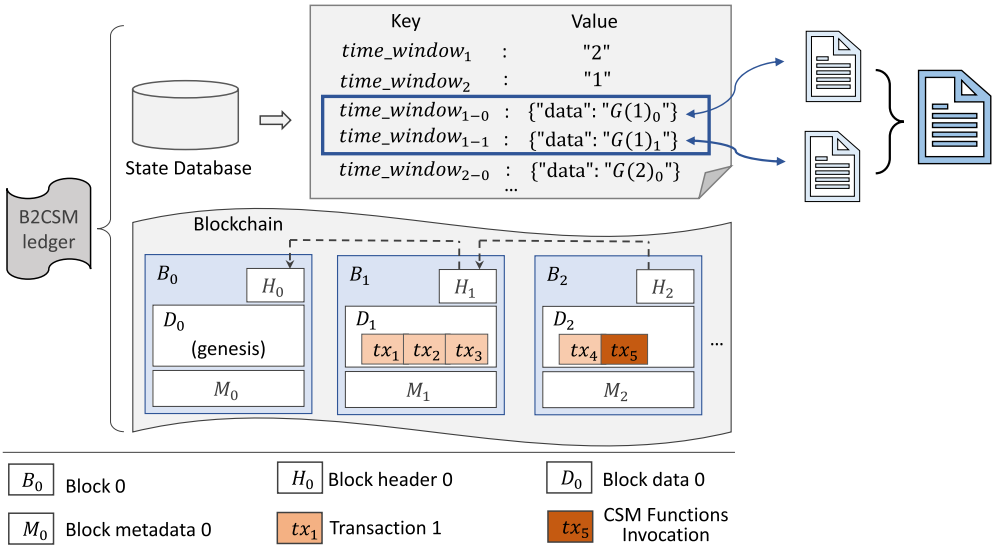


Fig. 9. Illustration of the B2CSM ledger structure with a blockchain and a state database. The state database stores the cyber data units.

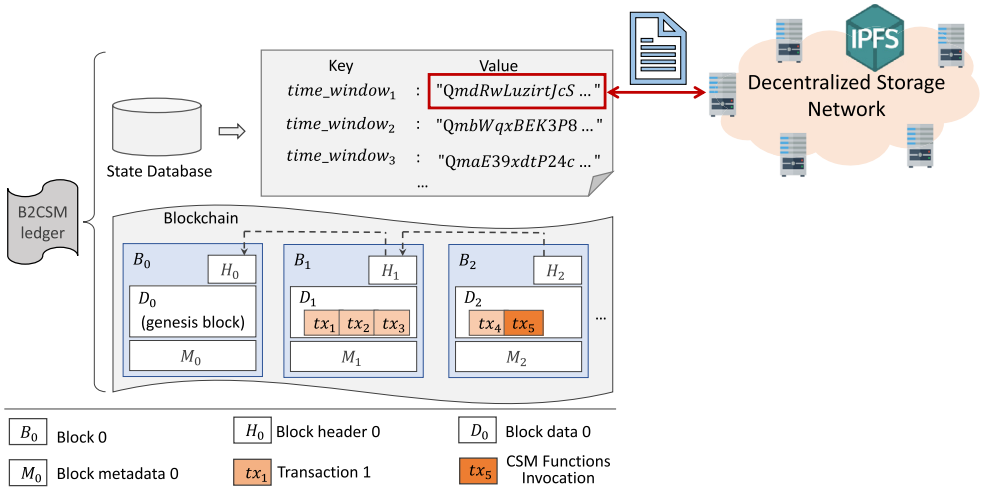


Fig. 10. Illustration of the B2CSM ledger structure with a blockchain and a state database. The state database stores content id *cid* of the cyber data that is returned by IPFS.

shown in Fig. 9, where a large $G(t)$ is divided into multiple data units. For example, $time\_window_1$ consists of two data units that are respectively keyed by $time\_window_{1\text{-}0}$ and $time\_window_{1\text{-}1}$. When defenders make CSM queries, the B2CSM middleware can invoke the CSM functionalities in the smart contracts, which take as input the relevant cyber data that is retrieved from the state database. This fine-grained ledger structure leverages the advantages of both blockchain and database structures [44] to facilitate blockchain-based applications as they process large volumes of data.

Method 2 ($\mathcal{M}_2$): Integrating with decentralized storage network. An alternative way of handling a large volume of cyber data is to incorporate a decentralized storage network, instantiated by IPFS. The key idea lies in storing the real cyber data in the IPFS while recording a reference, i.e., the returned content id *cid* of the cyber data on blockchain. We leverage the (private) IPFS cluster [31] that can be deployed by the defender on its own servers instead of the public version to store the cyber data since in that case, the data uploaded to IPFS typically needs to be encrypted, leading to extra cost for data decryption during retrieval.

One potential issue that may appear in the Fabric-IPFS enabled hybrid architecture is that a corrupted full node in the B2CSM blockchain network may maliciously modify or drop the *cid*, leading to subsequent inaccessibility. To tackle this issue, we leverage a gossip-based[1] diffusion method [22] and propose an augmented consensus mechanism. At a high-level, the replication procedure $\mathcal{M}_2$ works as follows: (i) a B2CSM agent signs the collected cyber data $G(t)$ along with the time window t and submitted to $f+1$ full nodes in the defender created private B2CSM blockchain channel; (ii) these $f+1$ full nodes execute the gossip procedure so that all full nodes can cache the cyber data temporarily; (iii) the *leader* node in the consensus mechanism submits the cyber data to IPFS and receives the *cid*, then starts the BFT consensus, e.g., via BFT-SMaRt [8] consensus mechanism, with other full nodes, eventually all honest full nodes receive the correct *cid*; (iv) each full node retrieves the data in IPFS using the *cid* and verifies the attached signature generated by defender, whereupon it updates the state database with the key of time window t and the value of *cid*, as depicted in Fig. 10, and then clears the locally cached cyber data. Otherwise, if the signature is invalid, a *view change* (VC) is triggered to elect a new leader node and restart from the prior

[1] A gossip protocol is a procedure where the cyber data $G(t)$ can be routed to all full nodes by letting each peer node randomly and uniformly select θ neighbor nodes and forward the data [21].

Listing 1 N-CSM cyber data $G(t)$ example (in state database).

```
{
  "timeWindow": "20211101", // it means "20211101-20211102" if replication frequency is one day
  "allDataUnits": [{
      "dataUnit": "20211101-0",
      "externalIPs": ["192.168.10.74", "192.168.10.75", "192.168.10.81"],
      "internalIPs": ["192.168.1.115", "192.168.1.116", "192.168.1.67"],
      "visitRecords": [["0","1","1"],["1", "1","0"],["0", "0","0"]]
    }, {
      "dataUnit": "20211101-1",
      "externalIPs": ["192.168.10.74", "192.168.10.75", "192.168.10.81"],
      "internalIPs": ["192.168.1.121", "192.168.1.124", "192.168.1.7"],
      "visitRecords": [["0","0","1"],["0", "1","1"],["1", "0","0"]] }, ...], ...
}
```

Listing 2 N-CSM cyber intelligence example.

```
{
"attackerIP": "192.168.10.74", // an external IP address
"timeInterval": "20211101-20211102"
}
```

step (iii). More concrete message flow and remarks are provided in Appendix B.1.

Comparison of the two methods $\mathcal{M}_1$ and $\mathcal{M}_2$. The method $\mathcal{M}_1$ stores the cyber data in the state database, which brings the advantage that the chaincode/smart contract of CSM functions can conveniently and efficiently retrieve cyber data for function execution. But for this method, it essentially stores the copy of the cyber data on each full node, leading to higher storage cost (compared with $\mathcal{M}_2$). For the method $\mathcal{M}_2$, the advantage lies in a lower on-chain storage cost, e.g., consider there are $\mathcal{N}$ full nodes in the channel and the cyber data size is 1 Gb, then the on-chain storage cost for $\mathcal{M}_2$ is $\mathcal{N} \times 46$ bytes (i.e., the length of a *cid*) along with the 1 Gb cyber data that stored in IPFS, while $\mathcal{N} \times$ 1Gb for method $\mathcal{M}_1$. However, the drawback of $\mathcal{M}_2$ becomes clear during data retrieval. Specifically, the CSM function needs to execute by taking as input the cyber data and threat intelligence, in that case, it is not common to let smart contract directly retrieve from IPFS (since it is external source which may cause non-determinism in Fabric chaincode). Hence, either new retrieval mechanism needs to be developed or else the query/invocation latency would become considerably large (for large cyber data) if the defender downloads from IPFS and feeds to the chaincode. How to get the best of these two methods will be an interesting question for future work.

Phase II: CSM functionality invocation. After the cyber data $G(t)$ is replicated to the Fabric channel (i.e., via method $\mathcal{M}_1$) or the Fabric-IPFS enabled architecture (i.e., via method $\mathcal{M}_2$), the defender Bob can invoke CSM functions to identify potential risks with a given piece of threat intelligence. We defer the concrete message flow to Appendix B.2 and describe the high-level idea here: (i) the defender submits the time window t, the CSM type (such as N-CSM), and the intelligence via their B2CSM App, which forwards these to multiple full nodes in the private channel; (ii) these full nodes (i.e., the B2CSM middleware on them) execute the CSM functions and sign the result; (iii) the B2CSM App aggregates the results from these full nodes and present to the defender.

Cyber threat intelligence sharing. Besides the two main phases above in B2CSM system, a potential additional phase is cyber threat intelligence sharing. Practically sharing cyber intelligence is done at the defender's discretion. In B2CSM, if the defender Bob would like to share cyber intelligence with other defenders, the following operations can be conducted: (i) the shared cyber intelligence can be encrypted using public key encryption (PKE) so that the sensitive information contained in the intelligence data can be kept confidential; (ii) additionally, the cyber intelligence data can be shared in a consortium channel, as discussed in Section 3.2.1, where only the defenders who would like to share with each other are involved and can access it. Such a consortium channel-based sharing mechanism brings an added advantage of accountability due to the immutability property of blockchain.

Also, in both cases, the defender who shares the (encrypted) cyber intelligence can sign the shared intelligence, and the resulting signature acts as the proof of authenticity of the intelligence data. As mentioned earlier, guaranteeing the authenticity of the threat intelligence *per se* is an orthogonal research problem and the extensive study of cyber intelligence sharing [3,5,11,13,41,42] in B2CSM system naturally forms one future work.

3.2.3. A specific CSM functionality invocation demonstrating data flow

Now we utilize a specific CSM function N.1 and the method $\mathcal{M}_1$ to demonstrate a concrete data flow. As a pre-execute phase, the cyber data for N.1 is collected and stored as an adjacency matrix (as discussed in Section 2.1.3) where each row represents an external IP address, each column is an internal IP address, and the value (either 1 or 0) in the i-th row and the j-th column indicates whether such an external IP has visited the internal IP during a time period or not. The adjacency matrix is parsed in JSON format and then submitted to the private channel that was created by the defender in phase I (i.e., Section 3.2.2). Listing 1 illustrates an example of stored cyber data (units) in state database.

Consider the example of function N.1, which aims to identify potential victims of an attacker with attackerIP during time interval $[t_1, t_2]$. In this case, the following steps occur: (i) the defender invokes the B2CSM App with the threat intelligence shown in Listing 2 and specifies that the channel is N-CSM; (ii) the App sends a request to multiple full nodes, and on each node the B2CSM middleware invokes the N.1 function that deployed as chaincode in the N-CSM channel; (iii) the chaincode retrieves the cyber data from state database and executes the predefined processing functions and outputs the potential victim IP addresses that have been attacked by attackerIP during time interval [11/01/2021, 11/02/2021]; (iv) the B2CSM middleware signs the output on behalf of the full node and returns the results along with signature to the App; (v) the server running the B2CSM App verifies the signatures for the results received from the full nodes and shows the defender a set of victims' IP addresses. Listing 3 further shows an example of the cyber data that stored in IPFS. Note that such a potentially large JSON file in Listing 3 is split into chunks of 256 KB and stored on different IPFS peer nodes.

Listing 3 N-CSM cyber data $G(t)$ example (in IPFS).

```
{
  "timeWindow": "20211101", // it means "20211101-20211102" if replication frequency is one day
  "cyberData": {
    "externalIPs": ["192.168.10.74", "192.168.10.75", "192.168.10.81",...],
    "internalIPs": ["192.168.1.115", "192.168.1.116", "192.168.1.67",...],
    "visitRecords": [["0","1","1",...],["1", "1","0",...],["0", "0","0",...],...]},
  "oracle_proof": { "value": "..." },
  "defender_signature": { "value": "..." },
  "meta_data": { "timestamp": "...", ... }
}
```

In the case of the crashing of some full nodes in the N-CSM channel, a defender is notified that those servers are unreachable. Furthermore, if the signature verifications of some full nodes fail, the defender will also be notified that those servers are suspected victims. Consequently, corresponding actions (e.g., replacing the suspicious full node and adding new servers to the N-CSM channel) can be taken by the defender. Note that the preceding discussion similarly applies to other CSM functions.

3.2.4. Security objectives

We define the following five security objectives for B2CSM:

Correctness. The correctness of the outputs of the CSM functions is assured, with respect to the input cyber intelligence and the cyber data $G(t)$.

Integrity. The integrity of data, namely the cyber data written by the B2CSM agents to the B2CSM blockchain network (and DSN in method $\mathcal{M}_2$), and the invocation history of the CSM functions stored in blockchain, is assured. This means the data cannot be manipulated without detection, as long as the fraction of compromised nodes in the underlying blockchain is bounded by a certain upper threshold.

Availability. The availability of the data stored in B2CSM is assured. Specifically, the cyber data written by the B2CSM agents to the B2CSM blockchain network (and DSN in method $\mathcal{M}_2$), and the invocation history of the CSM functions stored in the blockchain must remain available as long as the fraction of compromised nodes in the underlying blockchain network is bounded from above by a certain upper threshold.

Consistency. The consistency of the data, namely cyber data written by the B2CSM agents to the B2CSM blockchain network (and DSN in method $\mathcal{M}_2$), and the invocation history of the CSM functions stored in blockchain, is assured. This means all of the honest nodes in a B2CSM channel have the same global view about the data's state, as long as the fraction of compromised nodes in the underlying blockchain platform is bounded by a certain upper threshold.

Accountability. The B2CSM agents cannot write data into the blockchain network without record of the writing. Similarly, the B2CSM Apps cannot invoke CSM functions without record of the activities.

3.2.5. Threat model

We consider an attacker with the following capabilities: (i) The attacker can compromise B2CSM blockchain full nodes, by penetrating into some bounded fraction of them. The attacker has total control over these compromised nodes and can coordinate their activities in an arbitrary (i.e., Byzantine) fashion. (ii) The attacker can interfere with message deliveries. The attacker can control the order of message deliveries in the blockchain network. The attacker can arbitrarily delay message deliveries to each computer (but not forever, see Assumption 2 below) by waging Denial-of-Service (DoS) or other similar attacks. We consider the attacker with following standard abilities.

Assumption 1. Cryptographic assurance. We make standard assumptions to assure the security of cryptographic schemes (e.g., digital signatures). Informally speaking, these assumptions say that as long as cryptographic keys (if applicable) are not compromised, cryptographic schemes are secure. That is, in order for the attacker to compromise a cryptographic assurance, the attacker has to penetrate into a system in question to compromise the cryptographic keys or cryptographic service (for attaining "oracle" access to a cryptographic function) [53].

Assumption 2. Communication model. For the B2CSM blockchain network, we assume the communications between the full nodes are *partially synchronous*, meaning that each message is delivered to the honest nodes within some unknown delay [14]. While in other steps in the B2CSM system, the communication is considered *synchronous* in the sense that the message can only be delayed up to *a-priori* known time period Δ.

Assumption 3. Corruption threshold. For the full nodes in any channel (since each (private or consortium) channel represents a separated ledger) of the B2CSM blockchain network, we assume that no more than one-third of them are compromised simultaneously, which is inherent to the adopted Byzantine Fault-Tolerance (BFT) protocol [46].

We stress that the above assumptions 1, 2 and 3 are standard with respect to the underlying cryptographic primitives, network model, and consensus protocols.

3.2.6. Security analysis

Assume that the attacker cannot compromise a defender Bob or the computers running the B2CSM App since otherwise the attacker can manipulate the output arbitrarily. Then the security analysis of B2CSM systems instantiated from the B2CSM architecture is analyzed as follows. Note that the analysis is based on method $\mathcal{M}_1$, while extensive analysis for method $\mathcal{M}_2$ is deferred to Appendix C.

- The *correctness* states that the outputs of the CSM functions are reliable. To generate authentic outputs, we can analyze each step of execution during the whole data flow: (i) the authenticity of the input cyber intelligence is considered correct by validating the digital signature attached with the intelligence data, which is generated by the sharer; (ii) the integrity of $G(t)$ stored in B2CSM blockchain network can be ensured due to the immutability property of blockchain; (iii) with the authentic input cyber intelligence and integrated cyber data, the CSM functions can be correctly executed unless the attacker can manipulate the execution of smart contracts in blockchain, which is of negligible probability; and (iv) no more than one-third of the full nodes can be compromised simultaneously, namely assumption 3, which ensures that the defender will receive the correct outputs by picking the majority consensus (i.e., $f+1$ identical results or the majority of $2f+1$ returned results, where f is the number of malicious nodes that can be tolerated in the blockchain network with $\mathcal{N}$ full nodes [22], [12]) of the invocation results from the full nodes.

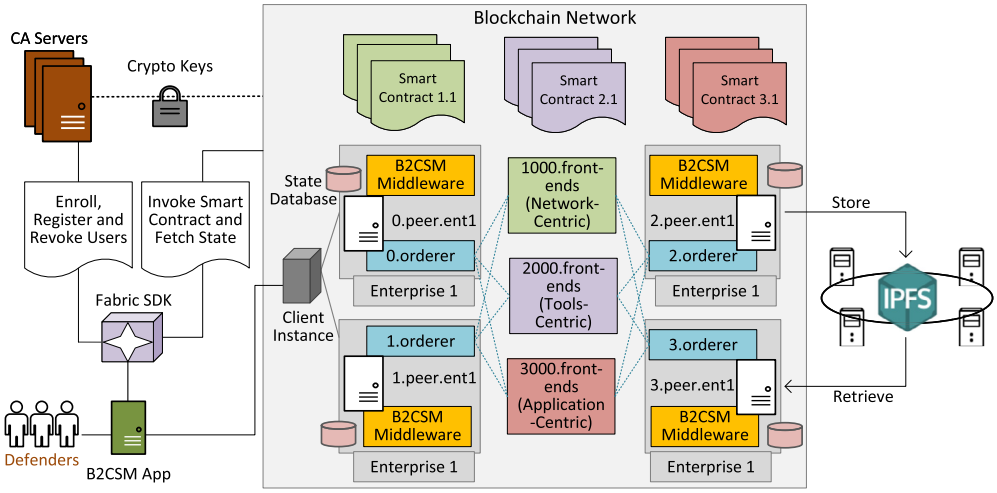


Fig. 11. Illustration of the B2CSM prototype system with 4 blockchain peer nodes, on which a private channel is created for one enterprise. Each node also acts as a replica of the BFT SMaRt-based ordering service and has a couchdb database. For method $\mathcal{M}_2$, the B2CSM middleware also contects with IPFS for cyber data writing and retrieval. The cases for 7 or 10 ordering (or peer) nodes follow the similar architecture and multiple-enterprise architecture can also be easily switched.

- The *integrity*, *availability* and *consistency* objectives are assured by the inherent properties of blockchain [6], including: (i) security of cryptographic primitives such as hash functions and digital signatures, namely assumption 1; (ii) the distributed architecture of the blockchain system; (iii) the execution of the consensus mechanism in the partial synchronous network model, i.e., assumption 2.
- The *accountability* is ensured as: (i) the data including B2CSM agents' public keys and timestamp are stored as transactions when writing cyber data to the blockchain network; (ii) when a defender invokes CSM functions, the smart contract is automatically triggered to record such an activity. Due to the aforementioned integrity of blockchain data, all the activities can be tracked, leading to accountability.

Overall, the CSM invocation can be automatically (due to the reliable execution of CSM functions in the pre-determined and deployed smart contract) performed without any manual inference, and the system is robust in the sense of all the aforementioned guaranteed security properties.

3.3. Analyzing B2CSM system performance

3.3.1. Performance metrics

We propose two CSM-specific performance metrics: *Data Replication Throughput* (DRT) and *Application Query Latency* (AQL). In particular,

The DRT metric measures the performance in writing data to the B2CSM blockchain. Since $G(t)$ is often large in volume and would be split into multiple chunks as in method $\mathcal{M}_1$, each with m rows and n columns, e.g., $m = 3$ and $n = 3$ in Listing 1. We call each chunk a *data unit*, whose size is limited by the transaction size in blockchain network. Let $|G(t)|$ be the size of $G(t)$ and $\mathcal{T}_{replication}$ be the total time cost for replicating $G(t)$ to the blockchain network. Then we define $\mathsf{DRT} = |\mathsf{G(t)}|/\mathcal{T}_{\mathsf{replication}}$.

The AQL metric measures the time interval between when a defender invokes a CSM function and when the defender receives the response, namely $\mathcal{T}_{invocation} = \mathcal{T}_{reqf} + \mathcal{T}_{cp} + \mathcal{T}_{resf}$, where $\mathcal{T}_{reqf}$ is the request formatting time (i.e., the time interval between the B2CSM middleware receiving a request from a B2CSM App and the B2CSM middleware submitting the transaction to the blockchain network), $\mathcal{T}_{cp}$ is the chaincode processing time (i.e., the time interval between the channel starting to execute the CSM function and the middleware receiving the query result from the blockchain network), and $\mathcal{T}_{resf}$ is the response formatting time (i.e., the time interval between the middleware receiving the result from the blockchain network and the middleware sending the result to the B2CSM App).

The above performance metrics are affected by the following *block-cutting* parameters that are involved when encapsulating transactions into blocks: *batch size* (by default, 10 transactions per block); *batch timeout* (by default, 2 seconds); and *block size* (by default, 512 KBytes). When the *batch size* or *block size* are met, or the *batch timeout* is reached, the OSNs encapsulate transactions into a new block. This means that one $G(t)$ might be stored into multiple blocks. Inspired by [49], we use the following block-cutting parameters in our experiments (unless explicitly specified otherwise): block timeout = 2 seconds; block size = 512 KB; batch size = 30 transactions per block.

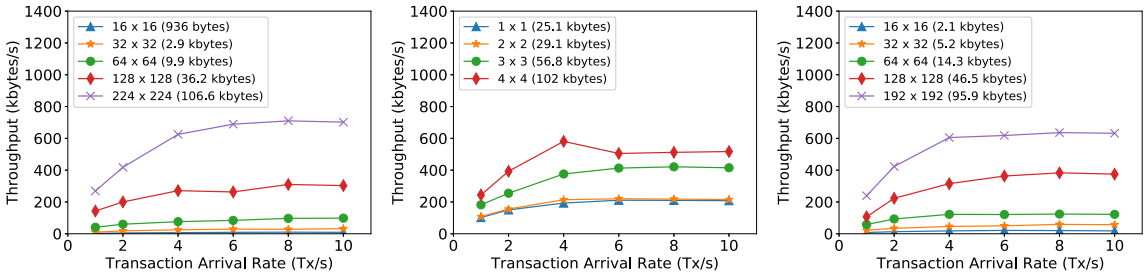
3.3.2. A B2CSM prototype system

We implement a prototype system of B2CSM to analyze the performance. The preceding design choices influence the prototype system, and a four-node architecture is depicted in Fig. 11. The B2CSM prototype system is built on top of a browser-server architecture. The B2CSM App has two modules: one displays blockchain-related information, including a dashboard with various kinds of information (e.g., B2CSM blockchain's peer nodes' IP addresses, the numbers of blocks and transactions for each channel). This presents a defender with the B2CSM blockchain's status in real-time. The other module offers a defender with a web-based interface to run the desired CSM functions with input cyber intelligence and receive the response from the CSM functions.

The Fabric software development kit provides the interfaces for interacting with the blockchain network (e.g., register users, install chaincode, instantiate chaincode, invoke transactions, and query ledgers). A Fabric client is instantiated when the defender initiates communication with the B2CSM blockchain network. This client only needs to be instantiated *once*, and subsequent sessions with the blockchain network can reuse it.

3.3.3. Experiments design and performance evaluation

We conduct experiments with the prototype system involving (as an example) one defender or enterprise, denoted by *ent1*. The defender has a range of CSMAs responsible for writing cyber data to the B2CSM blockchain network. The blockchain consists of four peer nodes, denoted by *0.peer.ent1*, *1.peer.ent1* and so on. These peer nodes are the full nodes for the B2CSM blockchain. There are



(a) N-CSM $\mathsf{DRT}_{\mathsf{CSM}}$ (b) T-CSM $\mathsf{DRT}_{\mathsf{CSM}}$ (c) A-CSM $\mathsf{DRT}_{\mathsf{CSM}}$

Fig. 12. B2CSM's $\mathsf{DRT}_{\mathsf{CSM}}$ in different CSM experiments (averaged over 5 independent runs).

four couchdb databases: *couchdb_peer0_ent1*, *couchdb_peer1_ent1*, etc. Each couchdb state database is connected with one peer node for recording its current world state.

There are four ordering nodes: *0.orderer*, *1.orderer*, *2.orderer*, and *3.orderer*, which act as the replicas for BFT SMaRt-based ordering service and it is assured that as long as the fraction of malicious nodes does not exceed 1/3 (i.e., 1 when there are 4 full nodes), the ordering service is secure. We also conduct the experiment on 7 (tolerating 2 faulty nodes) and 10 (tolerating 3 faulty nodes) ordering nodes that reside on peer nodes. There are three front-ends: *1000.frontends* (for N-CSM), *2000.frontends* (for T-CSM), and *3000.frontends* (for A-CSM). These front-end nodes are responsible for (i) relaying the transactions that are issued by the B2CSM clients to the consensus protocol and (ii) forwarding the blocks that are generated by the ordering nodes to peer nodes.

It is worth pointing out that the above architecture can be readily tuned to build a consortium blockchain network by re-running the network setup with changed configuration file such that, e.g., the peer nodes' names would change from *0.peer.ent1*, *1.peer.ent1*, *2.peer.ent1*, *3.peer.ent1* to *0.peer.ent1*, *0.peer.ent2*, *0.peer.ent3*, *0.peer.ent4* respectively and so do other service components such as ordering nodes, and then letting these components join in the same channel that a defender creates.

The hardware for conducting our experiments is a small-scale cluster of four Virtual Machines (VMs) residing on two heterogeneous servers, representing four nodes to formulate a private B2CSM blockchain. One server is a Dell PowerEdge R740, which is equipped with 2 Intel(R) Xeon(R) CPU Silver 4114 processors (with 13.75 MB L3 cache and 20 cores of 2.2 GHz for each processor), 256 GB (16 slots × 16 GB/slot) 2400 MHz DDR4 RDIMM memory, and an 8 TB (8 slots × 1 TB/slot) 2.5 inch SATA hard drive. The other server is a Dell Precision Rack 7910, which is equipped with 2 Intel(R) Xeon(R) CPU E5-2630 v3 processors (with 15 MB cache and 6 cores of 2.4 GHz for each processor), 16 GB 2133 MHz DDR4 RDIMM ECC memory, and a 256 GB 2.5 inch SATA solid state drive. The four VMs have the same configuration of 8 vCPUs, 24 GB memory and 800 GB hard drive and are connected via a Local Area Network (LAN). The operating system in each VM is Ubuntu 16.04 (64-bit) with kernel version 4.15. The Fabric version is 1.2, the Java version is 1.8.0_211, and the golang version is 1.11.10.

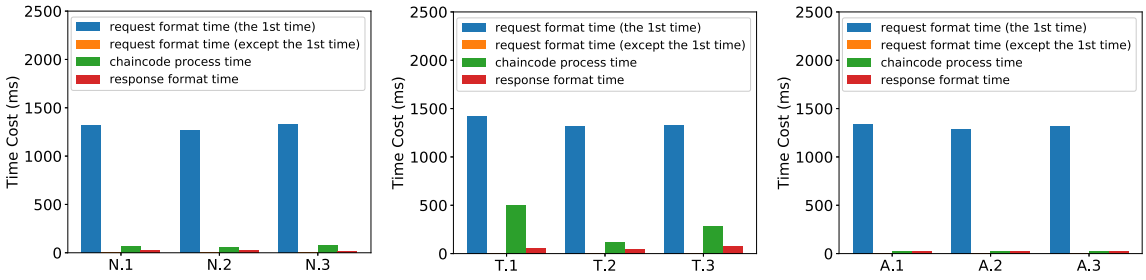
3.3.4. B2CSM performance based on experiments with real-world datasets

We now evaluate CSM-specific performance in DRT and AQL using real cyber data. In N-CSM experiments, we utilize a dataset collected from a honeypot during 7 days, and the time resolution is days (i.e., each day is a time interval). In T-CSM experiments, we use a dataset collected by the USMA team from the 2017 CDX Competition [38], as if it were collected at a production enterprise network, which indeed instantiates the model highlighted in Fig. 4. As this dataset does not have ground truth tags, for our experimental purposes, we replay the traffic using a popular open-sourced intrusion detection system, Suricata [47], with a popular, free ruleset referred to as Emerging Threats [15]. We store Suricata's alerts in an AGTSR $G(t)$ for time window t, where nodes represent the source and destination IP addresses of each attack. In A-CSM experiments, we consider the example of a defender recording how an enterprise's browsers have accessed the external URLs. In the simplest case, the cyber data is stored in the form $(browser, URL, timestamp)$, meaning that $browser$ accessed the URL at the time given by $timestamp$. Our experiments employ the Georgia Tech data received from [48] over the period of 2/1/2019-2/6/2019. The data contains mappings between malware instances, which are treated as browser applications for our purpose, and the external URLs. The data is pre-processed into a bipartite AGTSR over the time horizon of $T = 6$ days.

Figures (12a), (12b), and (12c) plot B2CSM's cyber data replication throughput (denoted by $\mathsf{DRT}_{\mathsf{CSM}}$) using the real-world datasets mentioned above. We observe that the throughput varies with CSM scenarios. The throughput of T-CSM is significantly different from those of N-CSM and A-CSM. This is caused by the fact that the T-CSM data is quite different from the N-CSM and A-CSM data as follows. The T-CSM data volume is large and the volumes of data units vary substantially where some data units contain more empty elements than others (recalling that T-CSM data is generated from network traffic); in contrast, N-CSM data and A-CSM data are uniformly distributed (i.e., data units are about the same size). This explains why T-CSM has a lower throughput. From the throughput, we observe that after the transaction arrival rate exceeds 4, the throughput stays stable, especially for N-CSM and A-CSM; this may be caused by the limited computing resources on the full nodes in our experiments. In T-CSM, we observe an "abnormal" throughput at transaction arrival rate 4 and data unit of 4×4 (i.e., 102 KBytes per unit); this may be caused by the limited computing resources at the full nodes and the cumulative effect of non-uniform distribution in the units' data volumes.

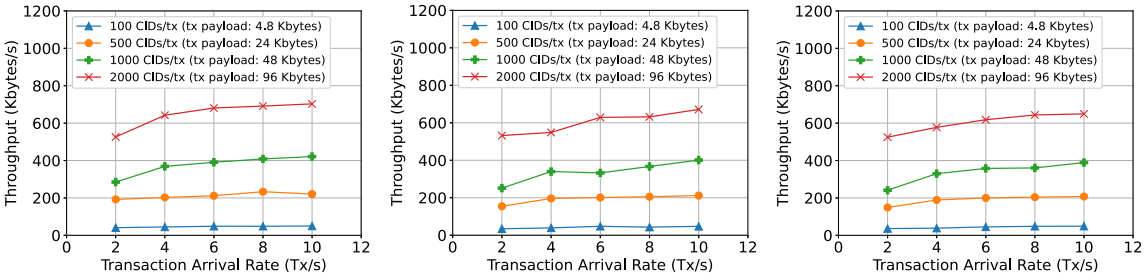
Figures (13a), (13b), and (13c) plot B2CSM's AQL using the real-world datasets mentioned above. We observe the following: (i) for the request formatting time, it takes about 1.4 seconds for the *first* invocation of a CSM function, but much smaller time for subsequent invocations. This is because the former requires us to initialize a (one-time) Fabric client object on behalf of the B2CSM App before connecting to the blockchain network; whereas, the latter can simply reuse the object created by the former. (ii) for the chaincode processing time, the time cost varies for different invocations of CSM functions. (iii) the response time is relatively stable (i.e., varies only slightly).

Table 1 further presents the break-down of the latency time, where $\mathcal{T}^1_{reqf}$ is the request formatting time when a CSM function is invoked *for the first time* by a B2CSM App and $\mathcal{T}^2_{reqf}$ is the request formatting time after the initial invocation of a CSM function. We highlight that the former time costs $\mathcal{T}^1_{reqf}$ is only one-time even though it is relatively longer. Besides, the chaincode processing



(a) N-CSM AQL (b) T-CSM AQL (c) A-CSM AQL

Fig. 13. B2CSM's AQL in different CSM cases (averaged over 5 independent runs).



(a) $\mathsf{DRT}_{\mathsf{bc}}$, 4 orderers and peers (tolerating $f = 1$ faulty node) (b) $\mathsf{DRT}_{\mathsf{bc}}$, 7 orderers and peers (tolerating $f = 2$ faulty nodes) (c) $\mathsf{DRT}_{\mathsf{bc}}$, 10 orderers and peers (tolerating $f = 3$ faulty nodes)

Fig. 14. B2CSM's $\mathsf{DRT}_{\mathsf{bc}}$ with different number of orderers (averaged over 5 independent runs).

Table 1
B2CSM's application query latency (unit: ms).

CSM classes	CSM functions	$\mathcal{T}^1_{reqf}$	$\mathcal{T}^2_{reqf}$	$\mathcal{T}_{cp}$	$\mathcal{T}_{resf}$
N-CSM	N.1	1321.11	0.17	69.18	23.47
	N.2	1265.16	0.18	57.6	23.49
	N.3	1329.34	0.17	75.86	18.37
T-CSM	T.1	1420.92	0.19	504.27	52.81
	T.2	1317.57	0.16	120.13	46.92
	T.3	1327.26	0.17	279.63	72.66
A-CSM	A.1	1336.33	0.21	28.92	28.14
	A.2	1287.17	0.19	27.51	24.23
	A.3	1324.84	0.17	30.33	30.62

time depends on the smart contract complexity (i.e., the complexity of a CSM function). Finally, the response formatting time $\mathcal{T}_{resf}$ is bigger than the request formatting time $\mathcal{T}^2_{reqf}$ when disregarding object-creating time during the first invocation of a CSM function; this is because each full node needs to sign the query results before sending them back to the B2CSM App. In summary, we have the following conclusion: *The response delay is mainly due to: (i) the creation of a Hyperledger Fabric client object corresponding to a CSM function invoked from a B2CSM App for the first time; and (ii) the specific chaincode execution of CSM functions. Reducing these time costs can correspondingly improve the response time.*

Scalability with varied number of nodes. Figures (14a), (14b), and (14c) plot the throughput (denoted by $\mathsf{DRT}_{\mathsf{bc}}$ since it is related to blockchain itself instead of specific CSM class) of replicating some general data (in the form of strings) such as (a batch of) content ids (cids[2]) to the blockchain network with 4 orderers (tolerating 1 faulty node), 7 orderers (tolerating 2 faulty nodes) and 10 orderers (tolerating 3 faulty nodes). Note that the ordering service is deployed on the peer nodes without delegating to extra nodes. Each transaction submitted to the blockchain network contains various numbers of content ids as payloads, which yields different transaction sizes and updates the state database via smart contract. The transaction arrival rate shows how many transactions are simultaneously submitted via multi-threads. Note that if we examine one specific CSM class, e.g., N-CSM, which possesses the same cyber data format, then the throughput $\mathsf{DRT}_{\mathsf{CSM}}$ follows the same pattern with $\mathsf{DRT}_{\mathsf{bc}}$ with respect to various number of orderer (or peer) nodes.

From the throughput $\mathsf{DRT}_{\mathsf{bc}}$, we have the following observations: (i) increasing the transaction size, namely incorporating more cids in a transaction, can significantly improve the throughput. However, the transaction in blockchain network has size limit for the sake of communication efficiency, e.g., once the payload size exceeds about 105 KB in our testing, the replication usually fails; (ii) with increased number of orderer nodes that can tolerate more faulty nodes, the throughput is slightly decreased. This is reasonable since more orderer nodes reaching consensus would cause more communication latency; (iii) the throughput can reach around 700 KB/s (or higher with more engineering optimizations) for replicating content ids to blockchain network. Though such a throughput is relatively slower than a distributed database-enabled system, e.g., the throughput for HBase is about 5 MB/s [1], yet the advantage lies in the robustness assurance, as characterized by the security properties analyzed in Section 3.2.6.

4. Limitations and future extensions

As the first step towards a fully automated and robust cyber security management system, the present study has several limitations which need to be addressed in future studies. First, the presented CSM classes (i.e., N-CSM, T-CSM and A-CSM) collectively do not yet cover all possible CSM functions. Future research needs to identify other CSM functions, which can be readily plugged into our proposed B2CSM system. Second, in N-CSM, the IP addresses

[2] We use cids here for all experiments since we are now examining the influence on the number of nodes, not data type; also, as we also consider that the cyber data to be replicated to IPFS, and only the returned content ids are stored in B2CSM blockchain network, the cyber data type (i.e., N-CSM, T-CSM or A-CSM) would not impact writing throughput to blockchain.

are used to identify attackers or victims; similarly, in A-CSM, URLs are used to identify potential victims. These identifiers may be easily to manipulate; for example, IP addresses may not be reliable when they can be spoofed or when attackers use anonymous communication tools. Hence, it is important to investigate more reliable identifiers. Third, the current design of B2CSM assumes that a piece of input cyber intelligence that is shared by other defenders is correct as long as the associated digital signature is valid. It is a challenging open problem to ensure the authenticity of cyber threat intelligence, especially when some defenders may have been compromised. Fourth, it would be useful to develop a visualization system to present the results of B2CSM. Fifth, it is of interest to integrate machine learning (as verifiable off-chain computation) with the smart contract execution to enhance the CSM functions.

5. Conclusion

In this work, we initiated the study of automated and robust cyber security management (CSM). This includes the formulation of three classes of CSM functions in relation to cyber threat intelligence sharing and a detailed description of the design of blockchain-based automated and robust CSM (B2CSM). We presented the implementation of a prototype B2CSM system. Experimental results based on real cyber datasets show that our system is useful in practice. We hope the limitations of our study will inspire more studies on this important problem.

CRediT authorship contribution statement

Songlin He: Investigation, Validation, Visualization, Writing – original draft, Writing – review & editing. **Eric Ficke:** Data curation, Methodology, Writing – original draft. **Mir Mehedi Ahsan Pritom:** Investigation, Validation. **Huashan Chen:** Data curation, Methodology, Software, Visualization. **Qiang Tang:** Conceptualization, Supervision, Visualization, Writing – original draft, Writing – review & editing. **Qian Chen:** Investigation, Validation, Visualization. **Marcus Pendleton:** Investigation, Validation. **Laurent Njilla:** Investigation, Validation. **Shouhuai Xu:** Conceptualization, Methodology, Supervision, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We thank the anonymous reviewers for their constructive comments. This work was supported in part by AFRL Grant #FA8750-19-1-0019, ARO Grant #W911NF-17-1-0566, and NSF Grant #1814825. Approved for Public Release; Distribution Unlimited. Case Number AFRL-2022-0399. Dated 28 Jan 2022.

Appendix A. CSM algorithms

A.1. N-CSM algorithms

Algorithm 1 realizes N-CSM function N.1 by identifying victims. The algorithm considers each time window within a given time interval $[t_1, t_2]$, checking each arc originating from the attacker to identify the entities that were accessed by the attacker. The query returns a list of all such entities. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_{t \in [t_1,t_2]} |V^{\text{internal}}(t)|)$, where $(t_2 - t_1 + 1)$ indicates the number of time windows that are considered.

Algorithm 1 N-CSM function N.1 (identifying victims).

Input: attacker, T, $G(t) = (V(t) = V^{\text{internal}}(t) \cup V^{\text{external}}(t), E(t), A(t))$ for $t \in [t_1, t_2]$ with $0 \le t_1 \le t_2 \le T$
Output: $\langle t, \text{victims}(t) \rangle$ for $t \in [t_1, t_2]$

for $t \in [t_1, t_2]$ **do**
  if attacker $\in V^{\text{external}}(t)$ **then**
    victims$(t) \leftarrow \emptyset$;
    for $v \in V^{\text{internal}}(t)$ **do** ▷ Check victims
      if $A_{\text{attacker},v}(t).count > 0$ **then**
        victims$(t) \leftarrow$ victims$(t) \cup \{v\}$;
return victims(t) for $t \in [t_1, t_2]$

Algorithm 2 N-CSM function N.2 (identifying potential attackers).

Input: victim_IP, T, $G(t) = (V(t) = V^{\text{internal}}(t) \cup V^{\text{external}}(t), E(t), A(t))$ for $t \in [t_1, t_2]$ with $0 \le t_1 \le t_2 \le T$
Output: $\langle t, \text{attackers}(t) \rangle$ for $t \in [t_1, t_2]$

for $t \in [t_1, t_2]$ **do**
  if victim_IP $\in V^{\text{internal}}(t)$ **then**
    attackers$(t) \leftarrow \emptyset$;
    for $a \in V^{\text{external}}(t)$ **do** ▷ Check attackers
      if $(a, \text{victim\_IP}) \in E(t)$ **then**
        attackers$(t) \leftarrow$ attackers$(t) \cup \{a\}$;
return attackers(t) for $t \in [t_1, t_2]$

Algorithm 3 N-CSM function N.3 (identifying extended victims).

Input: victim_IP, T, $G(t) = (V(t) = V^{\text{internal}}(t) \cup V^{\text{external}}(t), E(t), A(t))$ for $t \in [t_1, t_2]$ with $0 \le t_1 \le t_2 \le T$
Output: $\langle t, \text{potential\_victims}(t) \rangle$ for $t \in [t_1, t_2]$

for $t \in [t_1, t_2]$ **do**
  potential_victims$(t) \leftarrow \emptyset$;
  if victim_IP $\in V^{\text{internal}}(t)$ **then**
    tmp_attackers $\leftarrow \emptyset$;
    for $u \in V^{\text{external}}(t)$ **do**
      if $A_{u,\text{victim\_IP}}(t).\text{count} > 0$ **then**
        tmp_attackers$(t) \leftarrow$ tmp_attackers$(t) \cup \{u\}$; ▷ u accessed victim_IP
    for $u \in$ tmp_attackers(t) **do**
      for $v \in V^{\text{internal}}(t)$ **do**
        if $A_{u,v}(t).\text{count} > 0$ **then**
          potential_victims$(t) \leftarrow$ potential_victims$(t) \cup \{v\}$; ▷ u accessed v and may have compromised it
return potential_victims(t) for $t \in [t_1, t_2]$

Algorithm 2 realizes N-CSM function N.2 by identifying potential attackers based on their communications to a given victim. The algorithm considers each time window within the time interval $[t_1, t_2]$, checking which attacker entities tried to access the given victim entity. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_{t \in [t_1,t_2]} |V^{\text{external}}(t)|)$.

Algorithm 3 realizes N-CSM function N.3 by identifying potential victims that may be attacked by the attacker that caused the compromise of the input victim. The algorithm uses Algorithm 2 to compute the potential external attackers, which are then used to identify the other internal entities that may have been compromised by the potential attackers. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_{t \in [t_1,t_2]} |V^{\text{internal}}(t)| \cdot \max_{t \in [t_1,t_2]} |V^{\text{external}}(t)|)$.

A.2. T-CSM algorithms

Algorithm 4 realizes T-CSM function T.1 by inferring the attack paths to the compromised internal entity (e.g., computer or IP address, namely input I-2B) in time interval $[t_1, t_2]$. The algorithm creates a tree of potential attackers from the given compromised internal entity. The tree grows according to the relevant network activities, and adds new nodes when new attackers are identified. The resulting tree structure contains the target as the root, compromised internal entities as internal nodes, and all possible attackers as the leaves. Since the given compromised entity belongs to the internal LAN, the algorithm's search space originates in $G^{\text{D}-\text{L}}(t')$ and branches out within the network until all entities

Algorithm 4 T-CSM function T.1 (inferring attack paths).

Input: Victim_IP, T, $G(t) = (G^{\mathrm{I-D}}(t), G^{\mathrm{D-L}}(t), G^{\mathrm{L-I}}(t))$ for $t \in [t_1, t_2]$ with $0 \le t_1 \le t_2 \le T$
Output: Attack_Paths $= (V^{AP}, E^{AP}, A^{AP})$

$V^{AP} \leftarrow$ {Victim_IP}; $E^{AP} \leftarrow \emptyset$; $A^{AP} \leftarrow \emptyset$;
for $t = t_2$ downto t_1 **do**
  Node_Queue $\leftarrow$ New FIFO;
  while Node_Queue is not empty **do** ▷ Conduct BFT
    for Vertex $v \in V^{AP}$ **do**
      Node_Queue.ENQUEUE(v);
    Searched_Nodes $\leftarrow \emptyset$; Current_Node $\leftarrow$ Node_Queue.DEQUEUE();
    if Current_Node $\in V^{\mathrm{D-L}}(t)$ **then**
      for Vertex $v \in V^{\mathrm{D-L}}(t)$ **do**
        if $A^{\mathrm{D-L}}_{v,\text{Current\_Node}}(t)$.alerts $\neq \emptyset$ **then**
          if $v \notin V^{AP}$ **then**
            $V^{AP} \leftarrow V^{AP} \cup \{v\}$;
            for Vertex $v' \in V^{AP}$ **do** ▷ Initialize empty arcs to each existing nodes
              $A^{AP}_{v,v'}$.alerts $\leftarrow \emptyset$; $A^{AP}_{v',v}$.alerts $\leftarrow \emptyset$;
          $A^{AP}_{v,\text{Current\_Node}}$.alerts $\leftarrow A^{AP}_{v,\text{Current\_Node}}$.alerts $\cup A^{\mathrm{D-L}}_{v,\text{Current\_Node}}(t)$.alerts;
          if $v \notin$ Searched_Nodes $\cup$ Node_Queue **then**
            Node_Queue.ENQUEUE(v);
    Searched_Nodes $\leftarrow$ Searched_Nodes $\cup$ Current_Node;
  for sub $\in \{\mathrm{I-D}, \mathrm{L-I}\}$ **do**
    for Vertex $v \in V^{AP}$ **do**
      if $v \in V^{\mathrm{sub}}(t)$ **then**
        for Vertex $v' \in V^{\mathrm{sub}}(t)$ **do**
          if $A^{\mathrm{sub}}_{v',v}(t)$.alerts $\neq \emptyset$ **then**
            if $v' \notin V^{AP}$ **then**
              $V^{AP} \leftarrow V^{AP} \cup \{v'\}$;
              for Vertex $\hat{v} \in V^{AP}$ **do** ▷ Initialize empty arcs to existing nodes
                $A^{AP}_{\hat{v},v'}$.alerts $\leftarrow \emptyset$; $A^{AP}_{v',\hat{v}}$.alerts $\leftarrow \emptyset$;
            $A^{AP}_{v',v}$.alerts $\leftarrow A^{AP}_{v',v}$.alerts $\cup A^{\mathrm{sub}}_{v',v}(t)$.alerts;
return Attack_Paths $= (V^{AP}, E^{AP}, A^{AP})$

Algorithm 5 T-CSM function T.2 (identifying victims of zero-day attacks).

Input: Attack_Signature, T, $G(t) = \{G^{\mathrm{I-D}}(t), G^{\mathrm{D-L}}(t), G^{L-I}(t)\}$ for $t \in [t_1, t_2]$ with $0 \le t_1 \le t_2 \le T$
Output: $\langle t$, Matches(t)$\rangle$ where $t \in [t_1, t_2]$

Matches $\leftarrow$ New linked list of empty lists
for $t \in [t_1, t_2]$ **do**
  for sub $\in \{\mathrm{I-D}(t), \mathrm{D-L}(t), L-I(t)\}$ **do**
    for Vertex $v \in V^{\mathrm{sub}}(t)$ **do**
      for Vertex $v' \in V^{\mathrm{sub}}(t)$ **do**
        if Attack_Signature $\subseteq A^{\mathrm{sub}}_{v,v'}(t)$.alerts **then**
          Matches(t) $\leftarrow$ Matches(t) $\cup \{(v, v')\}$
return Matches(t) for $t \in [t_1, t_2]$

have been considered. Once the relevant $G^{\mathrm{D-L}}(\cdot)$'s have been exhausted, the algorithm checks both $G^{\mathrm{I-D}}(\cdot)$ and $G^{\mathrm{L-I}}(\cdot)$ to identify potential external attackers. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot ((\max_{t\in[t_1,t_2]} |V^{\mathrm{D-L}}(t)|)^2 + \max_{t\in[t_1,t_2]} |V^{\mathrm{I-D}}(t)| + \max_{t\in[t_1,t_2]} |V^{\mathrm{L-I}}(t)|) \cdot \max_{t\in[t_1,t_2]} |V(t)|)$.

Algorithm 5 realizes T-CSM function T.2 by retrospectively detecting victims of a zero-day attack during the past time windows prior to discovery of the zero-day attack (i.e., input I-3). The cyber intelligence may come in the form of an alert sequence from either an IDS' output or a previously unexplained anomaly. In either case, the defender needs to look at all previous IDS alerts to find matches. For this purpose, the algorithm traces back over the past time windows in between t_1 and t_2, by looking at each IDS alert in the set of arc annotations. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot ((\max_{t\in[t_1,t_2]} |V^{\mathrm{D-L}}(t)|)^2 + \max_{t\in[t_1,t_2]} |V^{\mathrm{I-D}}(t)| + \max_{t\in[t_1,t_2]} |V^{\mathrm{L-I}}(t)|) \cdot \max_{t\in[t_1,t_2]} |V|(t))$.

Algorithm 6 realizes T-CSM function T.3 by identifying the cascading damage of a given attacker (i.e., input I-1A or I-1B). The algorithm determines which entities were targeted by the given attacker, either directly or recursively. The algorithm has a time complexity $\mathcal{O}(t_2 - t_1 + 1 \cdot \max_{t\in[t_1,t_2]} |V(t)|^2)$.

Algorithm 6 T-CSM function T.3 (identifying cascading damage).

Input: Attacker_IP, T, $G(t) = (G^{\mathrm{I-D}}(t), G^{\mathrm{D-L}}(t), G^{L-I}(t))$ for $t \in [t_1, t_2]$ with $0 \le t_1 \le t_2 \le T$
Output: Damage_Graph $= (V^{DG}, E^{DG})$

$V^{DG} \leftarrow$ {Attacker_IP}; $E^{DG} \leftarrow \emptyset$; $A^{DG} \leftarrow \emptyset$;
for $t \in [t_1, t_2]$ **do**
  for sub $\in \{\mathrm{I-D}, \mathrm{L-I}\}$ **do** ▷ Check arcs which come from the Internet
    if Attacker_IP $\in V^{\mathrm{sub}}(t)$ **then**
      for Vertex $v \in V^{\mathrm{sub}}(t)$ **do**
        if $A^{\mathrm{sub}}_{\text{Attacker\_IP},v}(t)$.alerts $\neq \emptyset$ **then**
          if $v \notin V^{\mathrm{DG}}$ **then**
            $V^{DG} \leftarrow V^{DG} \cup \{v\}$
            for Vertex $v' \in V^{DG}$ **do** ▷ Initialize empty arcs to existing nodes
              $A^{DG}_{v,v'}$.alerts $\leftarrow \emptyset$; $A^{DG}_{v',v}$.alerts $\leftarrow \emptyset$;
          $A^{DG}_{\text{Attacker\_IP},v}$.alerts $\leftarrow A^{DG}_{\text{Attacker\_IP},v}$.alerts $\cup A^{\mathrm{sub}}_{\text{Attacker\_IP},v}(t)$.alerts
  for Vertex $v \in V^{DG}$ **do**
    Node_Queue.ENQUEUE(v)
  Searched_Nodes $\leftarrow \emptyset$
  while Node_Queue is not empty **do** ▷ Conduct BFT
    Current_Node $\leftarrow$ Node_Queue.DEQUEUE()
    if Current_Node $\in V^{\mathrm{D-L}}(t)$ **then**
      for Vertex $v \in V^{\mathrm{D-L}}(t)$ **do**
        if $A^{\mathrm{D-L}}_{\text{Current\_Node},v}(t)$.alerts $\neq \emptyset$ **then**
          if $v \notin V^{\mathrm{DG}}$ **then**
            $V^{DG} \leftarrow V^{DG} \cup \{v\}$
            for Vertex $v' \in V^{DG}$ **do** ▷ Initialize empty arcs to each existing node
              $A^{DG}_{v,v'}$.alerts $\leftarrow \emptyset$
              $A^{DG}_{v',v}$.alerts $\leftarrow \emptyset$
          $A^{DG}_{\text{Current\_Node},v}$.alerts $\leftarrow A^{DG}_{\text{Current\_Node},v}$.alerts $\cup A^{\mathrm{D-L}}_{\text{Current\_Node},v}(t)$.alerts
          if $v \notin$ Searched_Nodes $\cup$ Node_Queue **then**
            Node_Queue.ENQUEUE(v)
    Searched_Nodes $\leftarrow$ Searched_Nodes $\cup$ Current_Node
return Damage_Graph $= (V^{DG}, E^{DG}, A^{DG})$

Algorithm 7 A-CSM function A.1 (identifying compromised browsers).

Input: app_id, T, $G(t)$ for $t \in [t_1, t_2]$ and $0 \le t_1 \le t_2 \le T$
Output: $\langle t$, suspicious_app(t)$\rangle$ for $t \in [t_1, t_2]$

for $t \in [t_1, t_2]$ **do**
  suspicious_app(t) $\leftarrow \emptyset$
  temp_URL_set $\leftarrow \emptyset$
  for $v \in V^{\mathrm{URL}}(t)$ **do**
    if (app_id, v) $\in E(t)$ **then**
      temp_URL_set(t) $\leftarrow$ temp_URL_set(t) $\cup \{v\}$ ▷ v was accessed by app_id
  for $v \in$ temp_URL_set(t) **do**
    for $u \in V^{\mathrm{app}}(t)$ **do**
      if $(u, v) \in E(t)$ **then**
        suspicious_app(t) $\leftarrow$ suspicious_app(t) $\cup \{v\}$ ▷ app u accessed URL v and is therefore suspicious
return $\langle t$, suspicious_app(t)$\rangle$ for $t \in [t_1, t_2]$

Algorithm 8 A-CSM function A.2 (identifying victims of a malicious URL).

Input: url_id, T, $G(t)$ for $t \in [t_1, t_2]$ and $0 \le t_1 \le t_2 \le T$
Output: $\langle t$, victim_apps(t)$\rangle$ for $t \in [t_1, t_2]$

for $t \in [t_1, t_2]$ **do**
  victim_apps(t) $\leftarrow \emptyset$
  for $u \in V^{\mathrm{app}}(t)$ **do**
    if $E(t)[u,$ url_id$] \neq -1$ **then**
      victim_apps(t) $\leftarrow$ victim_apps(t) $\cup \{u\}$ ▷ Application u accessed url_id
return $\langle t$, victim_apps(t)$\rangle$ for $t \in [t_1, t_2]$

A.3. A-CSM algorithms

Algorithm 7 realizes A-CSM function A.1 by identifying suspicious internal applications (i.e., potentially compromised browsers). The input to the algorithm is a browser as an internal victim (i.e., input I-2B). The output is a set of compromised browsers (internal victims) that have accessed any URLs visited by the given compromised browser during time interval $[t_1, t_2]$. The time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_t |V^{\mathrm{app}}(t)| \cdot \max_t |V^{\mathrm{URL}}(t)|)$.

Algorithm 8 realizes A-CSM function A.2 by identifying victim browsers. The input to the algorithm is a known malicious URL (i.e., input I-1A). The output is the set of browsers (inter-

Algorithm 9 A-CSM function A.3 (identifying victim URLs and applications of spoofed URLs).

Input: url_id, T, $\tau_{distance}$, $G(t)$ for $t \in [t_1, t_2]$ with $0 \le t_1 \le t_2 \le T$
Output: $\langle t, \text{spoofed\_urls}(t), \text{victim\_apps}(t)\rangle$ for $t \in [t_1, t_2]$

1: **for** $t \in [t_1, t_2]$ **do**
2: spoofed_urls$(t) \leftarrow \emptyset$; victim_apps$(t) \leftarrow \emptyset$
3: **for** $v \in V^{\text{URL}}(t)$ **do**
4: **if** 0 <EDIT_DISTANCE(v, url_id) $\le \tau_{distance}$ **then** ▷ v spoofed the given URL url_id
5: spoofed_urls$(t) \leftarrow$ spoofed_urls$(t) \cup \{v\}$
6: **for** $v \in$ spoofed_urls(t) **do**
7: **for** $u \in V^{\text{app}}(t)$ **do**
8: **if** $(u, v) \in E(t)$ **then**
9: victim_apps$(t) \leftarrow$ victim_apps$(t) \cup \{u\}$
10: **return** $\langle t, \text{spoofed\_urls}(t), \text{victim\_apps}(t)\rangle$, $t \in [t_1, t_2]$

Algorithm 10 EDIT_DISTANCE(url$_1$, url$_2$) [34,50].

Input: url$_1$, url$_2$
Output: total_distance (edit distance between url$_1$ and url$_2$)

1: domain$_1 \leftarrow$ EXTRACTING_DOMAIN(url$_1$)
2: domain$_2 \leftarrow$ EXTRACTING_DOMAIN(url$_2$) ▷ extracting components; e.g., 'google' and 'com' for google.com
3: compo$_1 \leftarrow$ EXTRACTING_COMPO(domain$_1$)
4: compo$_2 \leftarrow$ EXTRACTING_COMPO(domain$_2$)
5: MAX $\leftarrow$ max(|compo$_1$|, |compo$_2$|)
6: MIN $\leftarrow$ min(|compo$_1$|, |compo$_2$|)
7: total_distance $\leftarrow 0$
8: **for** $i \leftarrow 0$ to MIN $- 1$ **do**
9: **if** |compo$_1$| >|compo$_2$| **then**
10: distance $\leftarrow$ LEVENSHTEIN(compo$_1$[MAX $- i$], compo$_2$[MIN $- i$])
11: **else**
12: distance $\leftarrow$ LEVENSHTEIN(compo$_2$[MAX $- i$], compo$_1$[MIN $- i$])
13: total_distance $\leftarrow$ total_distance + distance
14: **return** total_distance

nal victims) that accessed the malicious URL during time interval $[t_1, t_2]$. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_t |V^{\text{app}}(t)|)$, where $\max_t |V^{\text{app}}(t)|$ is the maximum number of browsers that accessed some URLs during a time window.

Algorithm 9 realizes A-CSM function A.3 by identifying victim browsers of spoofed (e.g., typo-squatted) URLs. The input to the algorithm is an abused URL url_id (i.e., input I-2A). The output is the set of possibly spoofed URLs, denoted by spoofed_urls(t), and the set of potential victim browsers, denoted by victim_apps(t), for $t \in [t_1, t_2]$. Lines 3-7 of Algorithm 9 find each of the spoofed URLs $v \in V^{\text{URL}}(t)$ that has an *edit distance* smaller than a given threshold $\tau\_distance$, where edit distance is computed using Algorithm 10 (which is a variant of the Levenshtein distance algorithm). Lines 1-2 of Algorithm 10 extract the domain names from url_1 and url_2. Lines 3-4 create the array of components (i.e., the components separated by the '.' character) for each of domain names. Lines 5-6 determine the maximum and minimum lengths of the component arrays respectively. Lines 8-16 compute the edit distance for each components of component$_1$ and component$_2$ starting from the last component (usually top-level domain names such as '.com' or '.net' are the last components) and sum the edit distances of individual components to get the total_distance between domain$_2$ and domain$_2$. For example, consider url$_1$ = "`mail.google.com/contact.php`" and url$_2$ = "`mali.g00gle.com/home.php`." We define their edit distance as the edit distance between domain$_1$ = `mail.google.com` and domain$_2$ = `mali.g00gle.com`. More specifically, it is the sum of the edit distance between components `mail` and `mali`, the edit distance between components `google` and `g00gle`, and the edit distance between components `com` and `com` respectively. Lines 9-15 of Algorithm 9 identify all the victim browsers that visited any of the spoofed URLs in set $spoofed\_urls(t)$. Algorithm 9 has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_t |V^{\text{app}}(t)| \cdot \max_t |V^{\text{URL}}(t)|)$.

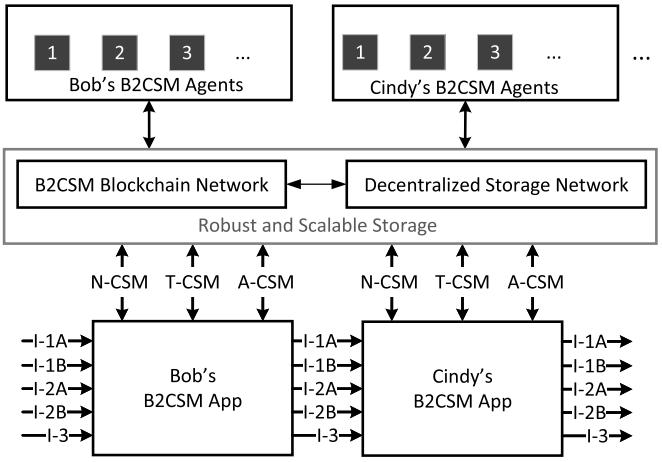


Fig. B.1. The B2CSM model extending from the CSM model using method $\mathcal{M}_2$.

Appendix B. B2CSM system design in Fabric-IPFS enabled architecture

Here we present the concrete message flow of B2CSM system in the Hyperledger Fabric and IPFS enabled architecture, i.e., the method $\mathcal{M}_2$. Fig. B.1 highlights the updated B2CSM model, which lends itself to the architecture shown in Fig. B.2. The notations are provided in Table B.1 for convenience of reference. First we introduce the cryptographic primitive of digital signature for the ease of later description.

Digital signature. We consider an *existential unforgeability under chosen message attack* (EU-CMA) secure digital signature scheme [18] SIG consisting of a tuple of algorithms (KeyGen, Sign, Verify) where:

- SIG.KeyGen$(\lambda) \rightarrow (pk, sk)$. The key generation algorithm takes as input the security parameter λ and outputs a pair of public key pk and secret key sk.
- SIG.Sign$(sk, m) \rightarrow \sigma$. The signing algorithm takes as input the secret key sk and the message m and produces the signature σ.
- SIG.Verify$(pk, m, \sigma) \rightarrow \{0, 1\}$. This deterministic verification algorithm takes as input the public key pk, the message m and the signature σ and outputs a boolean 1 or 0 indicating whether σ is valid on m relative to pk or not.

System setup. For a defender Bob, the following operations for system setup are executed: (i) a key pair $(sk^{\text{Bob}}, pk^{\text{Bob}})$ is generated via the signature scheme SIG; (ii) the full nodes provided by Bob in the B2CSM blockchain network form a committee $\mathcal{C}^{\text{Bob}}$ containing n^{Bob} $(n^{\text{Bob}} \ge 3f^{\text{Bob}} + 1))$ nodes $(\mathcal{C}_1^{\text{Bob}}, \cdots, \mathcal{C}_{n^{\text{Bob}}}^{\text{Bob}})$; each full node $\mathcal{C}_i^{\text{Bob}}$ also possesses a key pair $(sk^{\mathcal{C}_i^{\text{Bob}}}, pk^{\mathcal{C}_i^{\text{Bob}}})$ generated via the signature scheme SIG. All the secret keys are kept private while the public keys are publicly known. Meanwhile, Bob creates a private channel in the B2CSM blockchain network by letting the committee nodes $(\mathcal{C}_1^{\text{Bob}}, \cdots, \mathcal{C}_{n^{\text{Bob}}}^{\text{Bob}})$ join in the channel. For cyber intelligence sharing, if Bob agrees to share the cyber intelligence with another defender Cindy (or with more defenders), they would jointly create and let some of their full nodes join in a consortium channel. The consortium channel follows the same requirement that $n^{\text{consortium}} \ge 3f^{\text{consortium}} + 1$ where $n^{\text{consortium}}$ is the total number of full nodes in the consortium channel while $f^{\text{consortium}}$ is the maximum number of faulty nodes that can be tolerated.

Based on the setup, at a high-level, the B2CSM system operates in two main phases:

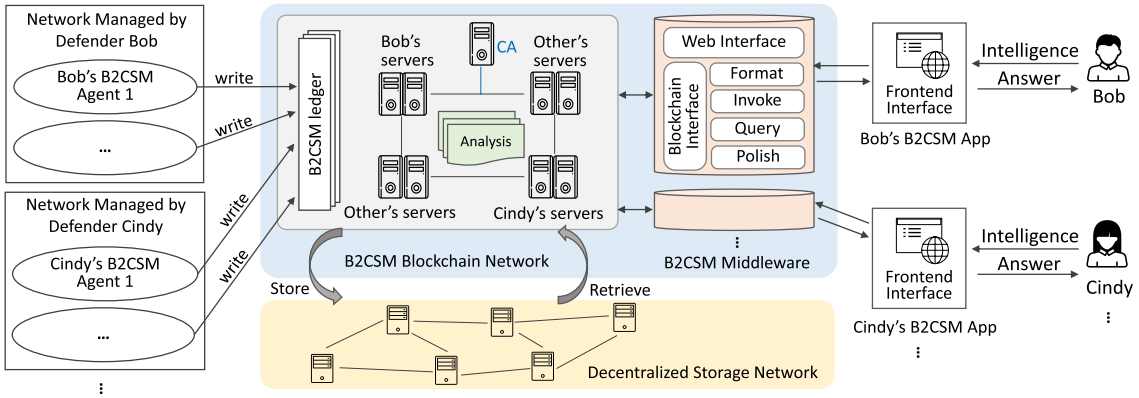


Fig. B.2. Illustration of B2CSM architecture based on Blockchain and decentralized storage network.

Table B.1
Key notations related to the $\mathcal{M}_2$-based B2CSM system design.

Notation	Represent for
$G(t)$	the collected cyber data for time window t by a B2CSM agent
$(sk^{\text{Bob}}, pk^{\text{Bob}})$	the defender Bob's secret key and public key
n^{Bob}	the number of full nodes provided by the enterprise managed by defender Bob
f^{Bob}	the number of faulty nodes that can be tolerated by the n^{Bob} full nodes
$\mathcal{C}^{\text{Bob}}$	the committee formed by the n^{Bob} full nodes provided by the defender Bob, it also interchangeably called the private *channel* or *ledger* created by defender Bob
$\mathcal{C}_i^{\text{Bob}}$	each full node in the committee $\mathcal{C}^{\text{Bob}}$, $i = \{1, \cdots, n^{\text{Bob}}\}$
$(sk^{\mathcal{C}_i^{\text{Bob}}}, pk^{\mathcal{C}_i^{\text{Bob}}})$	the secret and public key pair for the full node $\mathcal{C}_i^{\text{Bob}}$
θ	the parameter of selected neighbors in the gossip-based diffusion mechanism
$\mathcal{O}^{\text{oracle}}$	an oracle protocol providing a black-box call for the collected cyber data
sid	the session id used to uniquely identify a protocol instance
π^{oracle}	the authenticity proof generated by the oracle protocol $\mathcal{O}^{\text{oracle}}$
σ_{data}	the signature generated by the B2CSM agent on behalf of the defender Bob
cid	the content identifier returned by IPFS after uploading data to it
$\mathcal{T}$	a timer that ensures the consensus completes or else a view change will be triggered
p	the number of selected full nodes in channel $\mathcal{C}^{\text{Bob}}$ for CSM function invocation
res	the CSM invocation result; for party i, it is denoted with res_i

B.1. Phase I: cyber data replication

This phase allows a defender, e.g., Bob, to robustly store the cyber data via private channels in the B2CSM blockchain network. Upon the B2CSM blockchain network, the channels and the connection with IPFS are initialized, defenders can continuously write collected data to the channel via B2CSM agents. However, the following challenges arise:

Challenge 1: Ensuring integrity of replicated cyber data. In a Fabric-IPFS hybrid architecture, some nodes (even managed by one enterprise) may be corrupted and therefore act arbitrarily, such as dropping or tampering with the cyber data during replication. To guarantee the integrity, we leverage a *gossip-based diffusion mechanism* [22] and propose an *augmented consensus mechanism*.

Challenge 2: Guaranteeing authenticity of collected cyber data. Another technical challenge lies in ensuring the authenticity of collected cyber data. It is well-known that blockchain can guarantee the immutability of the data stored in the ledger, while cannot ensure the trustworthiness of the external data input, e.g., $G(t)$'s, submitted to blockchain. We stress that this is an orthogonal research problem about authentic data feeding to smart contract, which relates to the so-called *oracle protocols*.

In B2CSM, we consider a black-box call of feasible (centralized/decentralized) oracle protocols (denoted with $\mathcal{O}^{\text{oracle}}$) as building blocks such as trusted execution environment (TEE)-based [54], [10] or crowdsource-based [36]. The $\mathcal{O}^{\text{oracle}}$ takes as input some message m and outputs (m, π^{oracle}) where π^{oracle} is the proof of authenticity of m. For example, π^{oracle} can be instantiated by a signature $\sigma \leftarrow \mathsf{SIG.Sign}(sk_{\text{TEE}}, m)$, where sk_{TEE} is the secret key only known to a memory space (e.g., so-called *enclave* in *Intel Software Guard Extensions* (SGX)) [54] equipped on a B2CSM agent. The extension of instantiating oracle protocols for B2CSM cyber data forms an interesting future work.

Message flow of phase I. Now we present the concrete message flow of this cyber data replication phase. We omit the session id (denoted by *sid*) in the description, and it is trivial to add a unique *sid* for each session to defend against *replay attack*. As depicted in Fig. B.3, the message flow is as follows:

1) The cyber data $G(t)$ on a B2CSM agent that managed by the defender Bob is fetched by the oracle protocol $\mathcal{O}^{\text{oracle}}$, which outputs $(t, G(t), \pi^{\text{oracle}})$ where π^{oracle} is the authenticity proof over the time window t and the cyber data $G(t)$. The B2CSM agent signs the data $(t, G(t), \pi^{\text{oracle}})$ and obtains the signature $\sigma_{\text{data}} \leftarrow \mathsf{SIG.Sign}(sk^{\text{Bob}}, (t||G(t)||\pi^{\text{oracle}}))$. The message $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$ is then submitted to θ full nodes in the private channel formed by the committee nodes $\mathcal{C}^{\text{Bob}}$, where $\theta = f^{\text{Bob}} + 1$ and f^{Bob} is the number of faulty nodes that can be tolerated in $\mathcal{C}^{\text{Bob}}$, as shown in step (1) of Fig. B.3.
2) Upon receiving the message $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$, the θ full nodes would: (i) verify the signature σ_{data} via $\mathsf{SIG.Verify}(pk^{\text{Bob}}, (t||G(t)||\pi^{\text{oracle}}), \sigma_{\text{data}})$ and drop the message if σ_{data} is invalid, otherwise store in their local caches; (ii) start a timer $\mathcal{T}$; and (iii) follow the gossip-based diffusion mechanism to keep forwarding $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$ to other full nodes, i.e., step (2) in Fig. B.3. At the end of this step, all honest full nodes would receive the message. Note that at the end of this step the message $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$ locates in each node's cache instead of writing to the ledger.

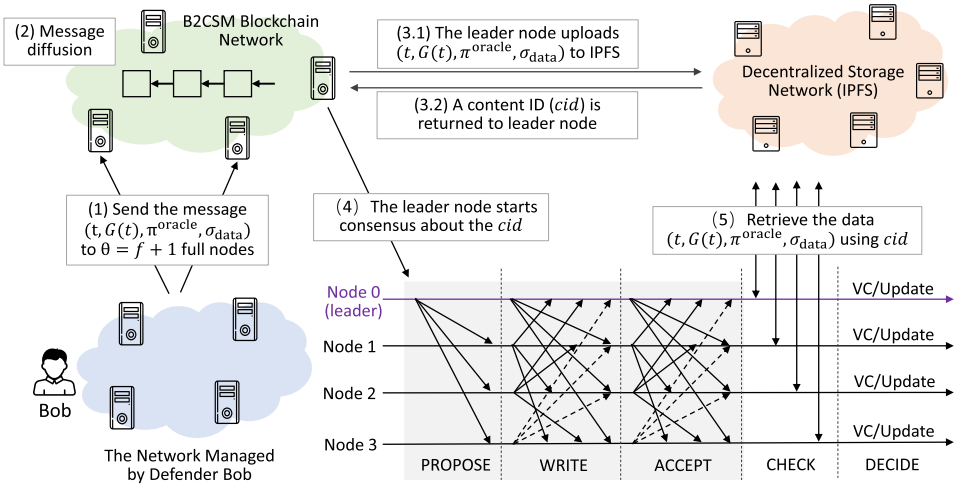


Fig. B.3. The message flow of the cyber data replication phase in B2CSM system.

3) The *leader* node (of the consensus mechanism) in Bob's private channel $\mathcal{C}^{\text{Bob}}$ starts to write the data $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$ to IPFS, which returns a content id (denoted by cid), i.e., steps (3.1) and (3.2) in Fig. B.3.
4) The leader node starts the consensus procedure for the cid, as shown in step (4) in Fig. B.3, where the PROPOSE, WRITE, ACCEPT are the rounds of the BFT-SMaRt consensus protocol [8].
5) Upon the consensus about cid completes, each full node utilizes the cid to retrieve the corresponding data $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$, i.e., the CHECK step in Fig. B.3, from IPFS and verify the signature σ_{data}. Then, as shown in the DECIDE step in Fig. B.3, if σ_{data} is valid, each honest full node updates the local ledger (i.e., the state database in Fabric) with the key of the time window t and the value of content id cid, exemplified by Fig. 10, and then clears the locally cached data. If σ_{data} is invalid, a *view change*[3] (VC) procedure is triggered to elect a new leader node and restart from step (3) in Fig. B.3.

Remarks. We have the following remarks about the steps above in phase I:

- Intuitively it is slightly more efficient if the full nodes $(\mathcal{C}_1^{\text{Bob}}, \cdots, \mathcal{C}_n^{\text{Bob}})$ immediately start to retrieve data from IPFS upon obtaining the cid, e.g., in the PROPOSE phase of the BFT-SMaRt protocol. However, in B2CSM, we propose to start the retrieval at the end of the consensus procedure to keep flexible plug-in of any BFT consensus protocols and facilitate efficient implementation and convenient deployment.
- Our design letting the leader node instead of all full nodes write the data $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$ to IPFS minimizes the communication complexity of interaction with IPFS.
- All full nodes in the private channel $\mathcal{C}^{\text{Bob}}$ only need to reach consensus on a short string of cid, e.g., 46 Bytes, instead of the large cyber data. While noting that temporarily caching the data $(t, G(t), \pi^{\text{oracle}}, \sigma_{\text{data}})$ on each full node $\mathcal{C}_i^{\text{Bob}}$, i.e., step (2) in Fig. B.3, is necessary due to the fact that each full node may be selected as a new leader to write the data to IPFS upon view change is triggered.
- During the CHECK step, multiple full nodes need to retrieve data from IPFS, which may pose some communication burden. However, such cost is acceptable since: (i) the connection between full nodes and IPFS peer nodes is typically long-lived; (ii) the number of full nodes connecting with IPFS is typically not large, e.g., in the magnitude of tens or hundreds; (iii) the request to IPFS is directed to multiple peer nodes in IPFS instead of a single one; (iv) the operation of cyber data replication relates to the specific writing frequency, e.g., in days.

B.2. Phase II: CSM functionality invocation

Given a piece of cyber intelligence, the CSM invocation functionality phase in B2CSM system operates as follows:

1) The defender Bob inputs the intelligence, the time window t of interest, and the CSM class (e.g., N-CSM) in B2CSM App, which submits the request to $p = f^{\text{Bob}} + 1$ full nodes in the channel $\mathcal{C}^{\text{Bob}}$ locating in B2CSM blockchain network.
2) The p full nodes retrieve the corresponding cyber data $G(t)$ from IPFS based on the time window t, and then send the cyber data together with the intelligence to CSM functions that deployed as smart contract in the $\mathcal{C}^{\text{Bob}}$ channel. The cyber data, if large, is split into chunks and temporarily cached in the state database in the $\mathcal{C}^{\text{Bob}}$ ledger.
3) The execution in smart contract is *deterministic* so that each of the p full nodes would in principle receive the same invocation result (denoted by res_i). Then each of them (i.e., the B2CSM middleware on it) executes $\text{SIG.Sign}(sk^{\mathcal{C}_i^{\text{Bob}}}, \text{res}_i)$ to obtain the signature σ_i and sends (res_i, σ_i) back to B2CSM App.
4) The B2CSM App executes $\text{SIG.Verify}(pk^{\mathcal{C}_i^{\text{Bob}}}, \text{res}_i, \sigma_i), i \in \{1, \cdots, p\}$ to check whether all the p signatures are valid and whether all $\text{res}_i, i \in \{1, \cdots, p\}$ are the same. If hold, the result res (which is any res_i since they are same) is treated as a valid final output. Otherwise, the B2CSM App can resubmit the request to $p = 2f^{\text{Bob}} + 1$ full nodes and then choose the majority from the returned p pairs as a final output.

Remarks. We have the following remarks for the steps above in phase II:

- An alternative way for CSM functionality invocation would be sending the request to all peer nodes in the channel $\mathcal{C}^{\text{Bob}}$ during the first step. Then at the last step, the B2CSM App waits for $f^{\text{Bob}} + 1$ pairs with valid signatures and of the same result, and

[3] If all $\mathcal{N}$ full nodes executing the consensus protocol have the same leader, they are in the same view. Views are numbered consecutively, and the leader of a view is a replica/peer p such that $p = v \bmod \mathcal{N}$, where v is the view number. A view change is carried out by setting the new leader to be $p = (v+1) \bmod \mathcal{N}$ to continue consensus execution [22].

output it as the final invocation result. Such a mechanism eliminates the possible re-submission described in the steps above yet may incur slightly more communication overload. The defenders can flexibly choose either way.

- To reduce verification complexity of multiple signatures from full nodes, two feasible extensions can be applied to this phase: (i) leverage multi-signature aggregation scheme [9] to aggregate the signatures; (ii) utilize non-interactive threshold signature [30] to let each full node generate a "partial" signature based on its secret key share; these partial signatures can be combined to a full signature for verification. Employing these two schemes can eventually produce only one signature (instead of multiple ones from full nodes) for efficient verification. But there need extra operations compared with the message flow we described above: for method (i), the B2CSM App needs to additionally perform *public key aggregation*; and for method (ii), it requires the execution of a distributed key generation algorithm [28] amongst the full nodes in $\mathcal{C}^{\mathsf{Bob}}$ during system setup. Properly integrating these methods into B2CSM for enhanced efficiency forms an interesting future extension.
- The cyber data caching time highly relates to the size of the caching data. To mitigate the possibly long latency, it is feasible to cache some cyber data in advance, e.g., those within one month, in the state database of the channel. Therefore, such a caching latency can be eliminated. Another possible way would be letting the smart contract/chaincode directly load cyber data from IPFS, in that case, however, any *non-determinism* in Fabric chaincode would lead to failure of CSM execution. Designing a mechanism to handle such a situation is an interesting extension.

Appendix C. Security analysis for Fabric-IPFS enabled architecture

We analyze that the security objectives (in Section 3.2.4) in B2CSM are satisfied in the Fabric and IPFS enabled architecture.

Correctness. The correctness states that the outputs of the CSM functions are reliable. We analyze each-step execution of the whole data flow:

- The authenticity of the input cyber intelligence is ensured by validating the digital signature attached with the intelligence data, which is generated by the sharer. We stress that ensuring the authenticity of the cyber intelligence *per se* is an orthogonal research problem.
- The integrity of cyber data $G(t)$ either during replication or stored in B2CSM blockchain and IPFS enabled hybrid architecture can be guaranteed, which is analyzed in later *integrity* property.
- With the authentic input cyber intelligence and integrated cyber data, the CSM functions can be correctly executed unless the attacker can manipulate the execution of smart contract in blockchain, which is of negligible probability.
- Since no more than one-third of the full nodes can be compromised simultaneously, namely assumption 3 in Section 3.2.5, our design in CSM functions invocation ensures the defender to receive the correct output via obtaining $f+1$ same results or the majority of $2f+1$ returned results, where f is the malicious nodes that can be tolerated in the specific blockchain channel.

Integrity. The integrity objective hinges on two aspects:

- *Integrity during replication*. During cyber data replication, we leverage a gossip based diffusion method and an augmented consensus mechanism to ensure the integrity. Specifically, there exist the following potential misbehaviors during replication: (i) the leader node may be corrupted and not send the data $(t, G(t), \pi^{\mathsf{oracle}}, \sigma_{\mathsf{data}})$ to IPFS or not start the consensus procedure after obtaining the *cid*; (ii) the leader node may be corrupted to maliciously modify the *cid* and try to let other full nodes reach consensus on the incorrect *cid*.

  Our B2CSM system handles these attacks by triggering a view change to elect a new leader node and restart from step (3) in Fig. B.3. Specifically, for case (i), the view change happens if more than $f^{\mathsf{Bob}}+1$ honest nodes do not receive *cid* after the timer $\mathcal{T}$ expires, and the duration of $\mathcal{T}$ is a system parameter; for case (ii), the view change is triggered by more than $f^{\mathsf{Bob}}+1$ honest nodes if they fail to verify the signature σ_{data} in the retrieved data $(t, G(t), \pi^{\mathsf{oracle}}, \sigma_{\mathsf{data}})$ from IPFS.
- *Integrity during storage*. The integrity of the stored cyber data $G(t)$ is assured due to the fact that: (i) the immutability property of B2CSM blockchain (which can be reduced to the security of cryptographic primitives such as hash functions and digital signatures, namely assumption 1 in Section 3.2.5) ensures the integrity of the content id *cid* of the cyber data; and (ii) IPFS ensures the integrity of the cyber data since the content id *cid* essentially is pertinent to the hash of the cyber data.

Availability. The availability objective is assured due to two perspectives: (i) the distributed architecture of blockchain and the corruption threshold (namely assumption 3 in Section 3.2.5) ensures that the blockchain network for querying the content id *cid* of the cyber data or invocation histories are always available; (ii) though by default the stored content in IPFS may fade away if no one access the content, we can still realize persistent storage by means of *pinning* in IPFS cluster [31], which excludes an object and its children from garbage collection (GC) within an IPFS node. Therefore, the cyber data stored in IPFS is always available.

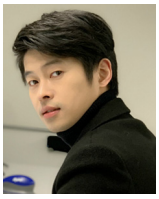
Consistency. The consistency objective is guaranteed by the *agreement* property [20] of the underlying consensus mechanism that operates in the partial synchronous network model in the blockchain network, i.e., assumption 2 in Section 3.2.5.

Accountability. The accountability objective is ensured due to the same reasons that described in Section 3.2.6.

References

[1] Apache, Hbase, https://hbase.apache.org/, 2021.
[2] M.T. Abdullah, S. Qidri, W. Nuryadi, S.R. Widianto, Failover cluster nodes and ISCSI storage area network on virtualization windows server 2016, J. Online Inform. (2020) 89–96.
[3] M.S. Abu, S.R. Selamat, A. Ariffin, R. Yusof, Cyber threat intelligence–issue and challenges, Indones. J. Elec. Eng. Comput. Sci. 10 (2018) 371–379.
[4] B.-T.S. Alliance, Galexia, Asia-Pacific cybersecurity dashboard - a path to a secure global cyberspace, http://www.bsa.org/APACcybersecurity, April 2015.
[5] P. Amthor, D. Fischer, W.E. Kühnhauser, D. Stelzer, Automated cyber threat sensing and responding: integrating threat intelligence into security-policy-controlled systems, in: Proc. of the 14th International Conference on Availability, Reliability and Security, ACM, 2019, pp. 1–10.
[6] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proc. of the 13th EuroSys Conference, ACM, 2018, pp. 1–15.
[7] J. Benet, IPFS-content addressed, versioned, p2p file system, preprint, arXiv: 1407.3561, 2014.
[8] A. Bessani, J. Sousa, E.E. Alchieri, State machine replication for the masses with BFT-smart, in: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2014, pp. 355–362.
[9] D. Boneh, M. Drijvers, G. Neven, Compact multi-signatures for smaller blockchains, in: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), Springer, 2018, pp. 435–464.
[10] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz, et al., Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks, Chainlink Labs, 2021.
[11] S. Brown, J. Gommers, O. Serrano, From cyber security information sharing to threat management, in: Proc. of the 2nd ACM Workshop on Information Sharing and Collaborative Security, ACM, 2015, pp. 43–49.

[12] M. Castro, B. Liskov, et al., Practical Byzantine Fault Tolerance, OSDI, vol. 99, Usenix, 1999, pp. 173–186.
[13] L. Dandurand, O.S. Serrano, Towards improved cyber security information sharing, in: 2013 5th International Conference on Cyber Conflict (CYCON 2013), IEEE, 2013, pp. 1–16.
[14] C. Dwork, N. Lynch, L. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM (JACM) 35 (1988) 288–323.
[15] Emergingthreats, Emerging threats rule server, https://rules.emergingthreats.net/, 2019.
[16] M.S. Ferdous, A. Margheri, F. Paci, M. Yang, V. Sassone, Decentralised runtime monitoring for access control systems in cloud federations, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2017, pp. 2632–2633.
[17] G. Fisk, C. Ardi, N. Pickett, J. Heidemann, M. Fisk, C. Papadopoulos, Privacy principles for sharing cyber security data, in: 2015 IEEE Security and Privacy Workshops, IEEE, 2015, pp. 193–197.
[18] S. Goldwasser, S. Micali, R.L. Rivest, A digital signature scheme secure against adaptive chosen-message attacks, SIAM J. Comput. (1988) 281–308.
[19] M. Gschwandtner, L. Demetz, M. Gander, R. Maier, Integrating threat intelligence to enhance an organization's information security management, in: Proc. of the 13th International Conference on Availability, Reliability and Security, 2018, pp. 1–8.
[20] B. Guo, Z. Lu, Q. Tang, J. Xu, Z. Zhang, Dumbo: faster asynchronous BFT protocols, in: Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2020, pp. 803–818.
[21] S. He, Q. Tang, C.Q. Wu, Censorship resistant decentralized IoT management systems, in: Proc. of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, 2018, pp. 454–459.
[22] S. He, Q. Tang, C.Q. Wu, X. Shen, Decentralizing IoT management systems using blockchain for censorship resistance, in: IEEE Transactions on Industrial Informatics (TII), 2019, pp. 715–727.
[23] S. He, Y. Lu, Q. Tang, G. Wang, C.Q. Wu, Fair peer-to-peer content delivery via blockchain, in: European Symposium on Research in Computer Security (ESORICS), Springer, 2021, pp. 348–369.
[24] U. Helmbrecht, S. Purser, G. Cooper, D. Ikonomou, L. Marinos, E. Ouzounis, M. Thorbrugge, A. Mitrakas, S. Capogrossi, Cybersecurity cooperation: defending the digital frontline, Tech. rep., ENISA, 2013.
[25] Hyperledger, The ordering service, https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html, 2020.
[26] International Organization for Standardization, ISO/IEC 27305:2016: information technology-security techniques-information security incident management, 2016.
[27] F. Jon, B. Mark, Definitive guide to cyber threat intelligence, Tech. rep., 2015.
[28] A. Kate, Y. Huang, I. Goldberg, Distributed key generation in the wild, IACR Cryptol. ePrint Arch. 2012/377, 2012.
[29] A. Kiayias, Q. Tang, Traitor deterring schemes: using bitcoin as collateral for digital content, in: Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2015, pp. 231–242.
[30] E. Kokoris Kogias, D. Malkhi, A. Spiegelman, Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures, in: Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2020, pp. 1751–1767.
[31] P. Labs, Ipfs cluster, https://cluster.ipfs.io/documentation/guides/pinning/, 2021.
[32] Y. Lu, Q. Tang, G. Wang, Zebralancer: private and anonymous crowdsourcing system atop open blockchain, in: 38th IEEE International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018, pp. 853–865.
[33] A. Marcella Jr, D. Menendez, Cyber Forensics: a Field Manual for Collecting, Examining, and Preserving Evidence of Computer Crimes, Auerbach Publications, 2007.
[34] F.P. Miller, A.F. Vandome, J. McBrewster, Levenshtein Distance: Information Theory, Computer Science, String (Computer Science), String Metric, Damerau–Levenshtein Distance, Spell Checker, Hamming Distance, Alphascript Publishing, 2009.
[35] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, 2008.
[36] K. Nelaturu, J. Adler, M. Merlini, R. Berryhill, N. Veira, Z. Poulos, A. Veneris, On public crowdsource-based mechanisms for a decentralized blockchain oracle, IEEE Trans. Eng. Manag. (TEM) (2020) 1444–1458.
[37] C. Patsonakis, K. Samari, M. Roussopoulos, A. Kiayias, Towards a smart contract-based, decentralized, public-key infrastructure, in: International Conference on Cryptology and Network Security, Springer, 2017, pp. 299–321.
[38] W.M. Petullo, B. Klimkowski, W.C. Moody, J. Bundt, M. Kranch, Cyber defense exercise artifacts, https://www.flyn.org/CDX/, 2017.
[39] D.B. Rawat, L. Njilla, K.A. Kwiat, C.A. Kamhoua, ishare: blockchain-based privacy-aware multi-agent information sharing games for cybersecurity, in: 2018 International Conference on Computing, Networking and Communications (ICNC), IEEE, 2018, pp. 425–431.
[40] M.N. Schmitt, Tallinn Manual 2.0 on the International Law Applicable to Cyber Operations, Cambridge University Press, 2017.
[41] O. Serrano, L. Dandurand, S. Brown, On the design of a cyber security data sharing system, in: Proc. of the 2014 ACM Workshop on Information Sharing & Collaborative Security, ACM, 2014, pp. 61–69.
[42] G. Settanni, Y. Shovgenya, F. Skopik, R. Graf, M. Wurzenberger, R. Fiedler, Acquiring cyber threat intelligence through security information correlation, in: 2017 3rd IEEE International Conference on Cybernetics (CYBCONF), IEEE, 2017, pp. 1–7.
[43] A. Shamir, How to share a secret, Commun. ACM (1979) 612–613.
[44] A. Sharma, F.M. Schuhknecht, D. Agrawal, J. Dittrich, Blurring the lines between blockchains and database systems: the case of hyperledger fabric, in: Proc. of International Conference on Management of Data (SIGMOD), 2019, pp. 105–122.
[45] F. Skopik, G. Settanni, R. Fiedler, A problem shared is a problem halved: a survey on the dimensions of collective cyber defense through security information sharing, Comput. Secur. 60 (2016) 154–176.
[46] J. Sousa, A. Bessani, M. Vukolic, A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform, in: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2018, pp. 51–58.
[47] Suricata, Open source IDS/IPS/NSM engine, https://suricata-ids.org/download/, 2018.
[48] G. Tech, GT malware HTTP daily feed 2018 dataset, https://www.impactcybertrust.org/dataset_view?idDataset=836, 2019.
[49] P. Thakkar, S. Nathan, B. Viswanathan, Performance benchmarking and optimizing hyperledger fabric blockchain platform, in: The 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2018, pp. 264–276.
[50] Wikipedia, Levenshtein distance, https://en.wikipedia.org/wiki/Levenshtein_distance, 2020.
[51] G. Wood, et al., Ethereum: a secure decentralised generalised transaction ledger, Ethereum project yellow paper, 2014, pp. 1–32.
[52] L. Xu, L. Chen, Z. Gao, X. Fan, K. Doan, S. Xu, W. Shi, Kcrs: a blockchain-based key compromise resilient signature system, in: International Conference on Blockchain and Trustworthy Systems, Springer, 2019, pp. 226–239.
[53] S. Xu, M. Yung, Expecting the unexpected: towards robust credential infrastructure, in: International Conference on Financial Cryptography and Data Security (FC), Springer, 2009, pp. 201–221.
[54] F. Zhang, E. Cecchetti, K. Croman, A. Juels, E. Shi, Town crier: an authenticated data feed for smart contracts, in: Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2016, pp. 270–282.



Songlin He is currently working towards the Ph.D. degree at New Jersey Institute of Technology, Newark, NJ, USA. His main research directions include Blockchain technology, security, privacy, and decentralized applications involving various application scenarios such as Internet of Things, Cyber Security, Content Delivery Networks, Decentralized Identities, etc. He is an IBM-certified Blockchain practitioner and instructor, and was a research scientist in Adobe Inc. He is also a member of IEEE and ACM.

Eric Ficke is pursuing a PhD in Computer Science at The University of Texas at San Antonio, where he received his Bachelor's degree in the same field. His research focus is in cyber security, including such topics as network monitoring & analysis, security metrics, and visualization.

Mir Mehedi Pritom is a Ph.D. student in Computer Science at The University of Texas at San Antonio. His research interests include on cybersecurity data analytics and quantification. He received his B.Sc. in Computer Science from the University of Dhaka, Bangladesh, and M.Sc. in Information Technology from University of North Carolina Charlotte.

Huashan Chen received the M.S. degree in computer science from the Institute of Information Engineering, Chinese Academy of Sciences in 2016, and the Ph.D. degree in computer science from the University of Texas at San Antonio in 2021. His primary research interests are in cybersecurity, especially moving target defense and security metrics.

Qiang Tang is currently a Senior Lecturer (US Associate Professor) at the University of Sydney. From 2016–2020, he was an assistant professor at New Jersey Institute of Technology and Director of JACOBI Blockchain Lab. He received his Ph.D from University of Connecticut in 2015 and did a postdoc at Cornell. His research spans broadly on applied/theoretical cryptography and blockchain technology. He received a few awards including MIT Technical Review 35 Chinese Innovators Under 35, Google Faculty Award and others. His Research has been supported by NSF, DoE, ARFL; Google, JD.com, and gifts from Protocol Labs and Stellar Foundation.

Guenevere (Qian) Chen (Member, IEEE) received a Ph.D. degree in Electrical and Computer Engineering from Mississippi State University, Mississippi State, MS, USA, in 2014. She is an Assistant Professor with the Department of Electrical and Computer Engineering, the University of Texas at San Antonio, San Antonio, TX, USA. Her primary research area is cyber security. Her research topics include human factors and their impacts on cybersecurity, blockchain, healthcare information system and IoMT security, intelligent transportation system security, industrial control systems security (SCADA and IIoT), software vulnerability detection, and end-to-end security solutions.

Marcus D. Pendleton is a Principal Research Engineer of Innovation for the 90th Cyberspace Operations Squadron, United States Air Force. Previously, he was a Research Engineer at the Air Force Research Laboratory (AFRL). His research interests include blockchain and cross domain technologies to cyber threat intelligence sharing. He earned his bachelor's degree at the State University of New York at Buffalo, and his Master's and PhD at The University of Texas at San Antonio.

Laurent L. Njilla received his B.S. in Computer Science from the University of Yaoundé 1 in Cameroon, the M.S. in Computer Engineering from the University of Central Florida (UCF) in 2005 and Ph.D. in Electrical Engineering from Florida International University (FIU). He is a Senior Research Electronics Engineer at the Information Assurance Branch of the U.S. Air Force Research Laboratory (AFRL), Rome, New York. Prior to joining the AFRL, he was a Senior Systems Analyst in the industry sector for more than 10 years. He is responsible for conducting basic research in the areas of hardware design, game theory applied to cyber security and cyber survivability, hardware Security, online social network, cyber threat information sharing, category theory, and blockchain technology.

Shouhuai Xu is the Gallogly Chair Professor in the Department of Computer Science, University of Colorado Colorado Springs (UCCS). He is the founding Director of the Laboratory for Cybersecurity Dynamics, which is driven by the systematic approach of Cybersecurity Dynamics to modeling and quantifying cybersecurity from a holistic perspective. This approach has three orthogonal research thrusts: metrics (for quantifying security, agility, resilience, risk and trustworthiness), cybersecurity data analytics, and cybersecurity first-principle modeling (for seeking cybersecurity laws). His research has been funded by AFOSR, AFRL, ARL, ARO, DOE, NSA, NSF and ONR. He co-initiated the International Conference on Science of Cyber Security (SciSec) and is serving as its Steering Committee Chair. He is/was an Associate Editor of IEEE Transactions on Dependable and Secure Computing (IEEE TDSC), IEEE Transactions on Information Forensics and Security (IEEE T-IFS), and IEEE Transactions on Network Science and Engineering (IEEE TNSE). He received a PhD in Computer Science from Fudan University in 2000. More information about his research can be found at https://xu-lab.org.