IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 4, 2021, accepted October 20, 2021, date of publication November 8, 2021, date of current version January 28, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3126401

Automated Responsible Disclosure of Security Vulnerabilities

ANDREA LISI[1,2], PRATEETI MUKHERJEE[3], LAURA DE SANTIS[4], LEI WU[3], DMITRIJ LAGUTIN[3], AND YKI KORTESNIEMI[3]

[1]Department of Computer Science, University of Pisa, 56127 Pisa, Italy
[2]Consiglio Nazionale delle Ricerche–IIT, 56124 Pisa, Italy
[3]Department of Communications and Networking, School of Electrical Engineering, Aalto University, 00076 Aalto, Finland
[4]Department of Industrial Engineering, University of Salerno, 84084 Fisciano, Italy

Corresponding author: Andrea Lisi (andrea.lisi@phd.unipi.it)

This work was supported by the EU Horizon 2020 Project SOFIE under Grant 779984.

ABSTRACT The disclosure of security vulnerabilities plays an important role in notifying vendors and the public about flaws in digital systems. Among the proposed disclosure approaches, the most utilized is Responsible Disclosure, which still suffers from several disadvantages such as fostering a false sense of security among the end-users, allowing arbitrary delays in the disclosure process, and forcing the party reporting a vulnerability to identify themselves, which has been exploited by vendors through intimidation and malpractice. To address these issues, this paper presents an improved version of the Responsible Disclosure approach called Automated Responsible Disclosure (ARD) - a solution that leverages distributed ledgers and interledger technologies to automate the disclosure process while offering increased security, privacy, and transparency. A prototype implementation has been released as open-source software, and the evaluation of the solution shows that ARD is capable of addressing the key shortcomings in existing solutions and fostering more transparent disclosure practices.

INDEX TERMS Responsible disclosure, automated responsible disclosure, security vulnerability, privacy, distributed ledger, interledger, smart contract, chaincode.

I. INTRODUCTION

Nowadays, when a significant portion of a person's life is dependent on digital systems, the security of those systems has become crucial due to the increasing number of cyber-attacks [1]. Alas, all digital systems suffer from vulnerabilities that expose them to attacks, and the vulnerabilities only increase as the systems become more complex [2]. The situation is further aggravated by the continuing trend of prioritizing time to market, particularly in the quickly expanding Internet of Things (IoT) market [3]–[5].

Unfortunately, despite extensive testing strategies, vulnerabilities are difficult to detect. To increase the efficiency of the vulnerability detection process, many companies offer rewards in the form of *bug bounty programs* in hopes of encouraging experts to actively look for vulnerabilities [6]. To manage the process of discovering, reporting, and patching vulnerabilities, different vulnerability disclosure approaches have been utilized over the years. The approaches are typically categorized as full vendor disclosure, full public disclosure, and responsible disclosure [7]–[10]: the *full vendor disclosure* approach restricts the communication of the vulnerability only to the vendor, which makes it too opaque for the public, the *full public disclosure method* favors warning the public, which may then also aid cyber-criminals, while the *responsible disclosure* approach tries to find a balance between the other two approaches by disclosing the vulnerability to the public only when a patch is released or when the time period to release a patch has expired. However, it still suffers from several problems. For example, since the experts are required to identify themselves, vendors are able to pressure them into concealing the vulnerabilities they discover from the public [11]–[15]. Moreover, the time period initially requested by a vendor to produce a patch can be relatively large and even arbitrarily extended during the process as the public is not aware of the vulnerability and is, therefore, not able to exert pressure for a timely patch release. Finally, another consequence of the public not being aware of the vulnerability is that they are not always able to

The associate editor coordinating the review of this manuscript and approving it for publication was Wen Chen.

take the most appropriate countermeasures until details of the vulnerability are publicized [16].

This paper presents a solution to these shortcomings by building on the idea of Automated Responsible Disclosure (ARD) proposed by Lagutin *et al.* [16]. The developed system leverages distributed ledgers, smart contracts, and interledger technologies to address the following research questions:

1) What are the trade-offs between transparency, information confidentiality, and privacy in the disclosure process (addressed in Section VII), and how can such requirements be balanced in a single solution (addressed in Section V)?
2) How does automation of the disclosure process affect the disclosure of vulnerabilities (addressed in Section VIII)?
3) How does the developed solution affect the latency and costs associated with the vulnerability disclosure process (addressed in Section VI)?

The contributions of this paper include (also compared to [16]):

- The idea of ARD has been developed into a detailed design where, for the first time, the transparency of disclosure, the confidentiality of vulnerability information, and expert's privacy are guaranteed simultaneously in a single solution;
- An open-source implementation of ARD built on Ethereum, Hyperledger Fabric, and the SOFIE Interledger component is presented;
- The solution is evaluated by quantitative analysis, including the costs and required number of transactions, and qualitative analysis with respect to a list of requirements and attack vectors.

The complete solution has been carefully designed to solve the issues of current responsible disclosure systems, while respecting the established practices of bounties and division of responsibilities. The prototype implementation and subsequent analysis confirm that the ARD system resolves the primary pain points in existing vulnerability disclosure processes, while being computationally efficient, resilient to attacks, and cost-effective.

The rest of the paper is organized as follows: Section II provides the background on security vulnerabilities, distributed ledgers, and interledger technologies; Section III describes the existing vulnerability disclosure approaches in more detail; Sections IV and V present the design and an open-source implementation of an ARD system, respectively; Section VI evaluates the system with quantitative measurements; Section VII qualitatively analyses the solution and Section VIII discusses the implications and future work; finally, Section IX concludes the paper.

II. BACKGROUND

This section introduces security vulnerabilities, distributed ledgers, and interledger technologies.

A. SECURITY VULNERABILITIES

The definition of a (security) vulnerability, according to the Common Vulnerabilities and Exposures (CVE), is *"A flaw in a software, firmware, hardware, or service component resulting from a weakness that can be exploited, causing a negative impact to the confidentiality, integrity, or availability of an impacted component or components"* [17]. In the European Union, the Cybersecurity Act [18] and the (NIS) Directive [19] establish the framework for regulating cybersecurity systems, while the ISO/IEC 29147:2018 [20] and ISO/IEC 30111:2019 [21] provide guidelines on receiving reports about potential vulnerabilities and terms of disclosure.

The coordination between different parties, such as the security experts and vendors, to report vulnerabilities is normally managed and encouraged through *Coordinated Vulnerability Disclosure* (CVD) policies and *bug bounty programs*. CVD aims to contribute to the security of IT systems by sharing knowledge about vulnerabilities [22], while vendor-sponsored bug bounty programs seek to incentivize the experts to hunt for vulnerabilities and define a strategy to report vulnerabilities in exchange for Vendor-defined rewards.

However, even with such mechanisms, security experts are not fully protected, and vulnerabilities are seldom addressed in a swift manner. An example of the former would be the case of the Australian hacker, Patrick Webster, who, in 2011, discovered a serious security flaw in First State's Superannuation System and, despite his ethical intentions, was accused of criminal acts and subjected to legal persecution [23]. A notable example for the latter concerns the well-known taxi service, Uber, which suffered from a security breach in 2016. The massive data breach resulted in granting unauthorized access to the credentials of over 57 million users worldwide, but the public disclosure of the breach happened only a year after the incident as the company attempted to cover up the situation by paying off the attacker [24].

The problems concerning the management of cybersecurity vulnerabilities are therefore as follows: i) *lack of transparency:* as evident from the aforementioned Uber case, transparency is essential to ensure awareness among the general public since, without any knowledge of the vulnerability, the general public is open to exploits; ii) *lack of privacy support:* while some experts wish to receive recognition for their services, many others may wish to preserve their privacy despite their involvement in the vulnerability disclosure process, and all want to avoid undue pressure from the vendor. However, current disclosure systems do not provide anonymity for the expert.

B. DISTRIBUTED LEDGER AND INTERLEDGER TECHNOLOGIES

A Distributed Ledger Technology (DLT) is a peer-to-peer (P2P) network, where every peer has access to an immutable shared history called ledger [25]. The peers agree, following a consensus protocol, on how to update the ledger by inserting

new records called transactions, but the records on the ledger cannot be modified by any party.

Several different DLTs have emerged over the years, with each system having a unique set of strengths and limitations depending on the architectural differences and design choices. According to the classification of Wust and Gervais [26], DLTs can be classified based on who is allowed to read the data and who is allowed to write new data. In a *public* DLT anyone can read the data, while in a *private* DLT only authorized parties have access to the data, so a public DLT provides transparency and non-repudiation of the data, while a private DLT provides privacy of the data. Similarly, in a *permissionless* DLT anyone can write data (provided they meet the requirements of the consensus mechanism), while in a *permissioned* DLT only authorized parties can perform writes. Of the possible four types of DLTs this classification enables, currently three types exist. Bitcoin [27] and Ethereum [28] are examples of public and permissionless DLTs, while Hyperledger Fabric [29] is a private and permissioned DLT. Moreover, public and permissioned DLTs such as Hyperledger Indy [30] exist, and they are well suited for cases where the DLT peers must be authorized because they need to be known, but public verifiability is also required.

In permissionless DLTs, maintaining a common state requires a complex consensus algorithm, which then makes it more secure from malicious modifications, while a permissioned DLT, given its more controlled setup, can maintain a common state with a much simpler consensus algorithm, thus making it more scalable and efficient but less secure than permissionless DLTs. These contradictory goals have been summarized in the *blockchain trilemma*, which states that a system based on a single DLT cannot be decentralized, secure, and scalable at the same time [31].

As no single DLT is able to satisfy all the requirements of every use case, interledger approaches [32]–[34] have been proposed to connect two or more DLTs to achieve a combination of features that enables, for example, linking together data stored in different DLTs, or transferring data from one DLT to another. Among the various interledger approaches, this work uses the *bridging approach* to transfer information between DLTs. In particular, the bridging approach enables one or two-way transfer of information between ledgers that are considered more or less equal [32].

Another key approach used in this work is hash-locking [34]: in a complex operation, some steps are locked with a *hashlock* until either someone provides the pre-image of the hashlock (at which time all the locked steps can take place), or the whole operation is canceled if a timer (called *timelock*) expires before the pre-image has been revealed. This technique can be implemented by Hash Time Locked Contracts (HTLCs) and is used, for example, in atomic cross-chain swaps [35].

III. VULNERABILITY DISCLOSURE

This section describes the previous work on the disclosure of vulnerabilities. Section III-A discusses the existing disclosure approaches and identifies their advantages and limitations, Section III-B covers the criteria a vulnerability disclosure approach should ideally satisfy, and finally, Section III-C summarizes the original idea of ARD.

The following terminology represents the actors involved in the disclosure process and shall be used throughout the rest of the paper [16]: *i)* security *Experts* discover vulnerabilities; *ii) Vendors* are the companies manufacturing the systems that may have vulnerabilities; *iii)* a consortium of *Authorities* manages the vulnerability disclosure processes e.g. for specific geographical locations, and *iv)* the *General Public* includes anyone interested in the security of the systems.

A. DISCLOSURE APPROACHES

According to the European Union Agency for cybersecurity (ENISA) [7], different approaches to the disclosure of vulnerabilities exist, but the details of the procedure, such as the submission form or a bounty system, may differ from Vendor to Vendor. Currently, there are three main approaches.

In the *Full Vendor Disclosure* approach, the existence of the vulnerability is communicated only to the Vendor, who is tasked with the responsibility of fixing it [8]. While this approach maintains secrecy in communication and prevents leak of vulnerability information to malicious parties [36], the general public is also kept unaware of both the vulnerability and the patching process, and the Vendor is not under any pressure to deliver a patch as fast as possible. Thus, this approach does not protect the users of the system if the Vendor acts irresponsibly and fails to patch the system within a reasonable time frame.

In the *Full Public Disclosure* approach the existence of the vulnerability is immediately communicated to the General Public, thus enabling the end-users to take the precautionary measures they deem appropriate. This approach also affects the Vendor since, depending on the exploit of the vulnerability, the Vendor could also be the target of an attack. Moreover, the Vendor risks potential loss of reputation that could result in irreparable marketing damages [9]. However, along with the General Public, malicious actors are also made aware of the vulnerability and they could exploit the issue before a patch is released.

The *Responsible Disclosure* approach is a trade-off between the aforementioned approaches and involves a trusted third party, such as the Computer Emergency Response Team/Coordination Center (CERT/CC), to coordinate the process. This party is responsible for making decisions in matters such as defining the conditions in which a vulnerability can or cannot be disclosed. In this approach, the existence of the vulnerability is communicated to the Vendor, but if the Vendor does not fix the vulnerability within the defined time frame (known as the grace period), the Expert is allowed to release the vulnerability to the General Public [10].

According to Cavusoglu *et al.*, none of the approaches is optimal but Responsible Disclosure is the approach that is most likely to ensure the release of a patch, and therefore, minimize the social loss, i.e., the investment in the patch and

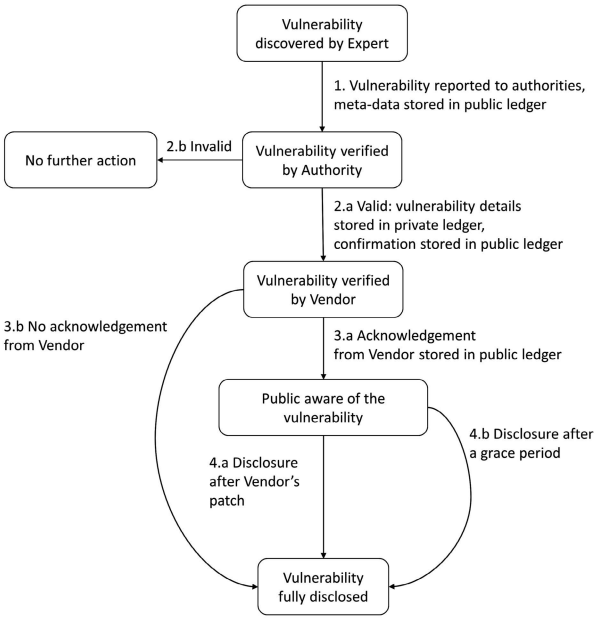


FIGURE 1. **The idea of ARD as proposed by Lagutin *et al.* [16].**

the damage caused by the vulnerability [37]. Arora *et al.* [38] perform a quantitative analysis on the CERT dataset and observe that the disclosure of a vulnerability doubles the likelihood of a patch release. However, Responsible Disclosure is opaque to the General Public, leaving open questions such as whether there is clear management of the deadlines, whether a patch entirely addresses the vulnerability, and whether there are existing vulnerabilities in the systems they use [16].

The UppsalaSecurity company proposed a blockchain-based platform called the *Sentinel Protocol* [39], which allows Experts to report attacks such as hacks, frauds, and scams, and a team of security experts, the Sentinels, then validates the reports and, if valid, publishes them on a blockchain that keeps a threat reputation database and rewards the related Experts. However, this solution does not involve any other party except the Experts and the Sentinel Protocol, making it unsuitable for the responsible disclosure of vulnerabilities, which involves the Vendor as well.

Hoffman *et al.* [40] propose a blockchain-based bug bounty program called *Bountychain*. Their application is powered by Ethereum smart contracts and the *Inter-Planetary File System* (IPFS) storage to build a safe, secure, and transparent platform for a bug bounty program by leveraging smart contract capabilities. Experts are allowed to submit vulnerabilities through the blockchain, where the acceptance process takes place. The payment process is automated via smart contracts. The transactions on the chain will serve as a persistent and transparent record of software bugs, while the bug details are recorded for long-term storage on IPFS. The system makes use of a database to store the vulnerability details during the acceptance process, but a submission is tracked only if accepted, and the Expert is unable to prove their submission in case of controversy. The Expert is also obligated to register to Bountychain as a tester and is, therefore, required to disclose their identity, which, as discussed earlier, makes them susceptible to threats and intimidation. Moreover, the solution does not address eventual modifications of the grace periods.

B. CRITERIA FOR VULNERABILITY DISCLOSURE

The overall goal of all the disclosure approaches is to make the available products more secure through transparency. Vendors that release more secure products (with fewer vulnerabilities) and are able to fix vulnerabilities in a timely manner, will gain reputation while vendors that fail to do so will lose it. Considering the upsides and downsides of each disclosure approach, it is possible to list the criteria, or requirements, a good disclosure approach should satisfy:

1) **Privacy**: the Expert is protected against intimidation through anonymity;
2) **Early disclosure**: if the Vendor does not acknowledge the vulnerability, the Expert is allowed to immediately disclose the vulnerability;
3) **Automation**: the disclosure approach should be as automated as possible, requiring only a minimal amount of manual intervention;
4) **Secrecy until disclosure**: the secrecy of the acknowledged vulnerability is preserved until disclosure;
5) **Transparency**: the information about the whole process of vulnerability disclosure is available to the General Public, including the number of vulnerabilities reported per Vendor and systems, number of accepted vulnerabilities, and their state in the resolution process.

C. THE IDEA OF AUTOMATED RESPONSIBLE DISCLOSURE

Lagutin *et al.* [16] proposed the idea of *Automated Responsible Disclosure (ARD)*, wherein the flow of information between Experts, Vendors, and Authorities is automated with the use of a public ledger, a private ledger, and interledger technologies. The basic process of ARD is summarized in Figure 1. An Expert discovers a vulnerability and communicates it to the Authorities (step 1). The Authorities evaluate the submission and, if approved, communicate it to the Vendor (step 2.a); otherwise, the Authorities reject the submission and the process ends (step 2.b). If the Vendor chooses to acknowledge the vulnerability (step 3.a), they promise to patch the vulnerability within a grace period; if they do not (step 3.b), either the Expert or the Authorities are allowed to disclose the vulnerability, which concludes the process. If the Vendor patches the vulnerability within the grace period (step 4.a), the ARD system automatically discloses the vulnerability and terminates the process; otherwise, if the grace period expires (step 4.b), either the Expert or the Authorities are allowed to disclose the vulnerability, leading to the completion of the procedure.

IV. THE ARD DESIGN

This section describes the design of a system based on the idea of Automated Responsible Disclosure (ARD), which

leverages both a public ledger for open access and a private ledger to address privacy concerns. The design highlights the key steps of the disclosure process, while in reality, the process can be more complex and may require additional dialogue between the parties in some of the exchanges. Support for such dialogue can, however, be added when deemed necessary.

A key feature of the design is the use of two ledgers and interconnecting them with an interledger technology. A public ledger is used for the status information that needs to be visible to everyone at all times to provide the necessary transparency. However, as the vulnerability details and related information need to be kept secret even from the other Vendors until disclosure, a public ledger alone does not suffice, so the design uses a private ledger for this information. This means the interledger technology connecting the two ledgers must support the *transfer of information* [32], i.e. vulnerability's related data and metadata. According to Siris *et al.* [32], the interledger approaches enabling the transfer of information are the bridging and the ledger-of-ledger approaches. The latter approach requires a complex setup with a main ledger validating the transactions of interconnected ledgers, which is not the focus of the ARD system. Therefore, the bridging approach is the most suitable and from now on, the term ''interledger'' will indicate the bridging interledger approach. With these key choices all 5 criteria can now be addressed successfully.

Using the same steps as in Figure 1 [16], Figure 2 illustrates the components of an ARD system, and how the participants interact with the system. A consortium of Authorities acts as an intermediary between Experts and Vendors by managing the disclosure of vulnerabilities. A Vendor V can register to the ARD system and delegate the responsibility of vetting the vulnerability reports to the consortium of Authorities. In exchange, V agrees to follow the protocol to acknowledge and patch vulnerabilities. Experts can then report vulnerabilities in a registered Vendors' products through a portal (e.g. a web page), which stores the status information to the public ledger (e.g. with Web3 technology [41]), while securely communicating sensitive information, such as the description of a vulnerability, directly to the Authorities. Key assumptions behind the design are 1) the Authority (ideally: a consortium of Authorities) is trustworthy and does not misbehave, and 2) the Vendor is solely responsible (to its customers and the public) for the fix of a vulnerability even if it internally uses (multiple) subcontractors.

To satisfy criterion 4, *secrecy until disclosure*, the design uses a private ledger, whose validator nodes belong to the consortium of Authorities and registered Vendors. Although the Vendor nodes participate in the private ledger, the private ledger must prevent a Vendor from accessing the data of the other Vendors, while the Authorities must have access to all the information. However, in a larger consortium of Authorities, a single Authority may not have access to every Vendor's data outside of the subset of Vendors it represents.

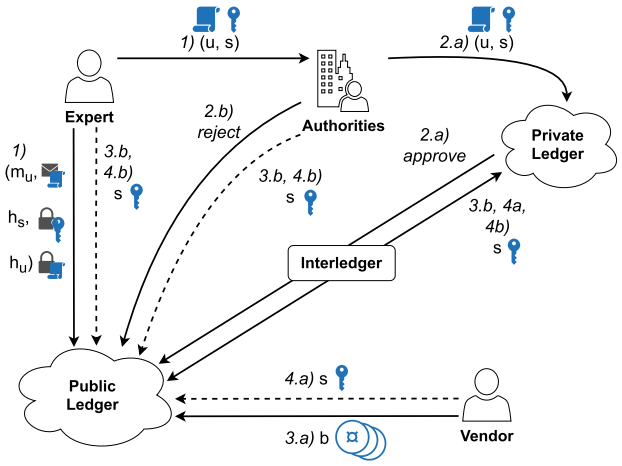


FIGURE 2. A high level view of the ARD system presented in this paper. The figure shows the components of the system (the private and public ledgers and the interledger) and how the participants interact with the system (symbols have been listed in Table 1). The step numbers match Figure 1. The dashed arrows 3.b, 4.a, and 4.b represent the operations to disclose a vulnerability and they are mutually exclusive (only one of them will be performed).

The public ledger can be any public distributed ledger secure enough to guarantee the immutability of data with a high level of confidence. Since the data is accessible to any peer, the public ledger should store only the information required to track the disclosure process and to prevent repudiation to satisfy criteria 2, *early disclosure*, and 5, *transparency*.

To protect the privacy of the Experts, they interact only with the Authority portal to report the vulnerabilities (possibly through suitable privacy-oriented tools) and sometimes with the public ledger[1] to trigger the disclosure of the vulnerability. Together, these measures satisfy the criterion 1, *privacy*.

The public ledgers are typically powered by cryptocurrencies, which can then be used to pay the bounties to the Experts. Even though the Expert might wish to receive the bounty immediately after their report has been approved by the Authorities, every single anonymous Expert cannot be trusted not to publish the vulnerability before a patch has been released. To discourage premature disclosures, in ARD the Expert receives the bounty only after the availability of a patch, and therefore after the official disclosure. Also, if there is reasonable doubt (which is deployment dependent) that the Expert has, nevertheless, published the vulnerability prematurely, the Authority is able to cancel the bounty promised to the Expert.

Finally, the interledger component is used to connect the two ledgers, and it is involved whenever a step of the process requires the interaction between public and private ledgers. This way, the trigger is only one transaction on one of the

[1]Most popular ledgers, such as Bitcoin end Ethereum, provide pseudonymity.

TABLE 1. The parameters used in the ARD process.

Symbol	Description
E	Expert
V	Vendor
A	Authority
P	Product
u	Vulnerability
m_u	Vulnerability metadata
s	Secret
$H()$	Hash function
h_s	Hashlock = H(s)
h_u	Vulnerability fingerprint = H(u)
t_a	Timelock to acknowledge
t_p	Timelock to patch
b	Bounty

ledgers, and interledger automatically manages the rest as required by the criterion 3, *automation*.

The following steps (illustrated in Figures 1 and 2) define the detailed ARD process (the symbols have been summarized in Table 1):

0) **Discovery:** An Expert E discovers a vulnerability u in Vendor V's product P. u represents the detailed information of the vulnerability, while m_u consists of the metadata of the vulnerability (product name and version, vulnerability type,[2] date of discovery, etc.). The Expert generates a random secret s and with hash function $H()$ computes the hashlock $h_s = H(s)$ and the hash of the vulnerability $h_u = H(u)$. h_u can then be used to verify u's integrity in later stages, for example by the Authorities at step 2.a, or by the General Public after the disclosure;

1) **Report:** E reports the new vulnerability u. The data E needs to submit is divided into public (the tuple (m_u, h_s, h_u)) and private (the tuple (u, s)) parts. Using, e.g., a portal run by the Authorities, E sends the public tuple to the public ledger while the private tuple goes to the Authorities for approval;

2.a) **Approval:** One or more of the Authorities $A \in$ *Authorities* verifies the integrity of the information, i.e. that $H(s) == h_s$, $H(u) == h_u$ etc. The main duty of A is then to determine (to the best of A's knowledge) whether u is a valid vulnerability, i.e. if it fulfills the conditions a) u can be reproduced; and b) it is a new discovery, not something already discovered by someone else. If A considers u a valid vulnerability, A stores the tuple (u, s) in the private ledger in such a way the tuple is shared with the Vendor V only (criterion 4, *secrecy*). However, to satisfy criterion 5, *transparency*, A is also required to communicate the approval of u to the public ledger, which is automatically handled by interledger using the information stored in the private ledger. As a consequence, the public ledger computes two timelocks, t_a and t_p, indicating the times in which V has to first acknowledge and then patch u;

2.b) **Rejection:** If $H(s) \neq s$, or $H(u) \neq u$, or u is not a valid vulnerability, A rejects E's submission. Then the only action performed by A is to store a flag, *Invalid* or *Duplicated*, in the public ledger to indicate that the vulnerability submitted by E is not a valid one. No information of this rejected vulnerability report is, therefore, recorded on the private ledger;

3.a) **Acknowledgment:** V now has time t_a to acknowledge on the public ledger their intention to fix u. To acknowledge, V stores a tuple $(Acknowledged, b)$ on the public ledger, where $b \geq 0$ is the bounty, in cryptocurrency, that V is willing to pay to E after the disclosure of u.

3.b) **Disclosure after the expiration of t_a:** If t_a expires, either E or A can disclose u. The disclosure is triggered by revealing s to the public ledger: if $H(s) = h_s$, an interledger operation reads u from the private ledger and stores it in the public ledger. If u is large, and the fee to store it is too large, an alternative solution is to store the location (e.g. a URL) of u;

4.a) **Disclosure after a patch:** V has time t_p to publish a patch to u. In order to do that, V stores the patch data, or a link to a location, e.g. a URL, if the patch data is too large, and the secret s on the public ledger. If $H(s) = h_s$, this triggers the disclosure process as explained in step 3.b. In this scenario, the location containing the vulnerability may also include the patch. If a reward was promised to E in Step 3.a, E gets rewarded within the same transaction;

4.b) **Disclosure after the expiration of t_p:** If t_p expires, E or A are allowed to disclose u by storing s to the public ledger and, if $H(s) = h_s$, this triggers the Disclosure step as described in Step 3.b. Similar to step 4.a, E gets rewarded within the same transaction.

V. THE ARD IMPLEMENTATION

This section presents a prototype implementing the ARD design and answers the second half of the research question 1) *"How can such requirements[3] be balanced in a single solution?"*. The implementation is available as open-source software [42] and is built on top of Ethereum [28] and Hyperledger Fabric [29] as the public and private ledgers, respectively, and utilizes the SOFIE Interledger component [43] for automating operations across the ledgers.

The ARD implementation uses Ethereum and Hyperledger Fabric DLTs because the design requires the public ledger to execute code, which is supported by Ethereum smart contracts, and it also requires the private ledger to store data that is hidden from some of the participants, which is supported by a combination of Private Data Collections and Channels in Fabric. Moreover, Ethereum and Fabric enjoy popularity in the community and of the availability of developer tools. Finally, the SOFIE Interledger component has been chosen because it implements the bridging approach supporting ledger agnostic communication with no assumptions on

[2]Denial of Service, Memory Overflow, Injection, etc., https://www.cvedetails.com/vulnerabilities-by-types.php

[3]Transparency, information confidentiality, and privacy.

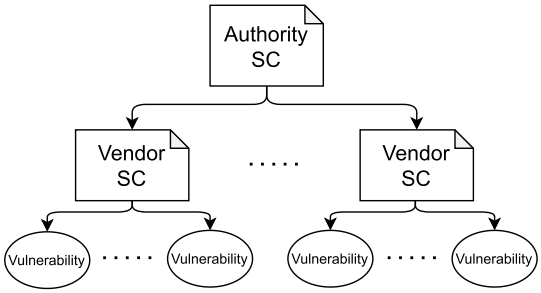


FIGURE 3. Relationship of the smart contracts on the public ledger. A circle indicates an internal data structure.

TABLE 2. The public fields representing a vulnerability in `Vendor SC`.

Field	Type	Size
Expert address	address	20 bytes
Hashlock (h_s)	bytes32	32 bytes
Secret (s)	uint256	32 bytes
Timelock (t_p)	uint32	4 bytes
Timelock (t_a)	uint32	4 bytes
State flag	uint8	1 byte
Bounty state	uint8	1 byte
Bounty amount (b)	uint256	32 bytes
Product ID	bytes32	32 bytes
Vulnerability hash (h_u)	bytes32	32 bytes
Creation timestamp	uint32	4 bytes
Disclosure data	string	X bytes
Total size		**194 bytes + X**

the ledgers, and it enables transactional information transfer for customized applications not limited to cryptocurrency assets [44]. However, the ARD design is not tied to Ethereum, Hyperledger Fabric, or the SOFIE Interledger component, and it can be deployed on any other ledger and interledger technology that supports the necessary functionalities.

The operations in the public ledger are implemented with Solidity smart contracts as explained in Section V-A while the operations on the private ledger are implemented with chaincode in Typescript for Hyperledger Fabric v1.4 as explained in Section V-B. Finally, Section V-C describes how the interledger operations have been implemented with the Interledger component. For simplicity, this implementation supports only one Authority though in real deployments a consortium of Authorities is recommended due to the lower trust requirements.

A. PUBLIC LEDGER SMART CONTRACTS

Interactions with Ethereum smart contracts require an Ethereum address. To simplify the descriptions below, the following addresses are assumed: *0xauthority* for the Authority, *0xvendor* for the Vendor, *0xexpert* for the Expert, and *0xinterledger* for Interledger.

The prototype builds on two smart contracts (SC) on the public ledger: one called `Authority SC` and another called `Vendor SC` as shown in Figure 3. The `Authority SC` is owned by the Authority, and it implements an HTLC contract (see Section II-B) to secure the operations. The contract stores the references to `Vendor SCs` belonging to Vendors registered in the ARD system with this Authority. A `Vendor SC` is owned by a Vendor, and it stores a data structure containing the public information of the vulnerabilities in that Vendor's products. The contracts provide the functions to proceed through the ARD process as described in Section IV, except for step 0 (Discovery), which is not controlled by the ARD and is assumed to be done independently by the Expert.

The `Authority SC` is called in several of the steps in the ARD process, and it provides the following functions:

- *registerVendor(vendor_address)* accepts new Vendor registrations and deploys the corresponding `Vendor SC` with *vendor_address* as its owner. This approach is more rigid than allowing a Vendor to implement and deploy a smart contract by themselves, but it reduces the possibility a Vendor attaches a vulnerable or malicious smart contract;
- *registerVulnerability(vendor_address, h_s, p_{id}, h_u)* implements the step 1, the submission of a new vulnerability involving the product with ID p_{id}. The parameter h_u will be used as unique identifier for u in the later steps;
- *reject(h_u, motivation)* implements the step 2.b, the rejection of a vulnerability identified by h_u with *motivation* indicating the motivation of the rejection i.e. a duplicate or invalid submission;
- *publishSecret(h_u, s)* implements steps 3.b, 4.a, and 4.b, beginning the disclosure of a vulnerability identified by h_u by exposing the secret s;
- *interledgerReceive(nonce, payload)* is a function the Interledger component [45] invokes to approve a vulnerability at step 2.a, and to terminate the disclosure at steps 3.b, 4.a, and 4.b.

The `Vendor SC` is responsible for storing the IDs of the products sold by the Vendor and the list of vulnerabilities reported by Experts. The contract exploits the *acknowledge(h_u, b)* function to trigger step 3.a, the acknowledgment of the vulnerability identified by h_u and assigning a bounty amount $b \geq 0$. The contract is also involved in every step that requires the `Authority SC` to store or update information about a vulnerability. The `Vendor SC` can be funded with cryptocurrency by the Vendor to also manage a bounty system that rewards the Experts who report approved vulnerabilities. However, the `Authority SC` operations do not involve any cryptocurrency so the contract never receives funds.

In the public ledger, each vulnerability u is represented by the fields listed in Table 2. The fields include the public variables listed in Table 1: the address of the Expert who submitted the vulnerability report, a state flag to track the disclosure process of each vulnerability, the bounty information that includes the amount and its state, the ID of the involved product, the timestamp of the submission, and the data containing the disclosure information.

In Ethereum, storing large amounts of data has a high cost in terms of gas (see Section VI-A), and, as a consequence,

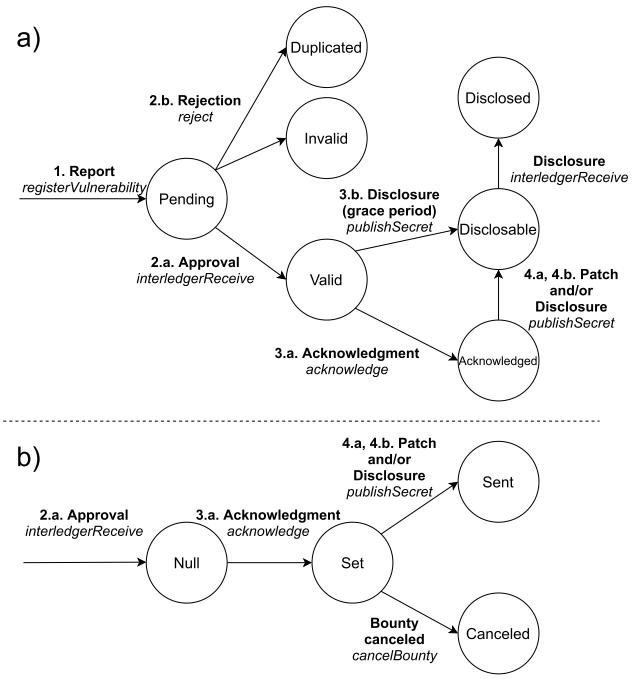


FIGURE 4. The state machines of a) the vulnerability and b) the bounty (bold indicates the steps of the procedure, *italic* the smart contract functions).

a large fee. Since the full vulnerability and patch descriptions can be very large, the prototype makes use of the following simplifications: (i) the disclosure data is a single string representing the URL of a public web page storing the vulnerability and patch descriptions (whose integrity can be verified since the vulnerability hash is available in the public ledger); (ii) the presence (or absence) of a patch is represented by a boolean flag emitted as an event by `Authority SC` and set to *true* if the secret is published by the Vendor (step 4.a) and *false* otherwise (steps 3.a and 4.b).

Possible states of the vulnerabilities and their evolution are shown in Figure 4 (a). The Figure shows in bold the actions as described in Section IV, and in italic the smart contract functions. A newly submitted vulnerability u is in *Pending* state. If u is approved by the Authority, u transitions to *Valid* state, otherwise either to *Duplicated* or *Invalid* state depending on the Authority's decision. When the Vendor acknowledges u, the vulnerability transitions to *Acknowledged* state. Finally, the disclosure requires two state transitions, to *Disclosable* when the secret is published, and *Disclosed* when the procedure is over and u has been disclosed.

A similar approach is used (and shown in Figure 4 (b)) to track the bounty for the Expert: when a vulnerability is approved, the bounty is initially in state *Null* until the Vendor acknowledges it. Once acknowledged, an amount $b \geq 0$ is set in cryptocurrency, and the state is updated to *Set*; when the vulnerability is in *Disclosable* state, the amount b is sent to the Expert's address, *0xexpert*, and the state is updated to *Sent*; the Authority has the power to cancel the bounty (in case the Expert has prematurely leaked the vulnerability) with the *cancelBounty*(h_u) function by setting b to 0 and the state to *Canceled*. However, this does not affect the disclosure of the vulnerability. Finally, a *Canceled* bounty cannot return to *Set* state.

B. CHAINCODE IN THE PRIVATE LEDGER

The private ledger exploits the concept of Private Data Collections (PDCs) to store sensitive vulnerability information in a secure manner. The collection definitions specified during the creation of private assets (in this case, vulnerability records) ensure that only the Vendor in question and the Authority have read/write access to the Collection. In particular, to safeguard confidential vulnerability details, Interledger does not function as a peer in the private ledger so it has no direct access to any data on the private ledger, but once it knows the secret matching the hashlock (either from the Expert, the Vendor, or the Authority in either steps 3.b, 4.a or 4.b), Interledger can request the vulnerability data from the application chaincode.

A separate Private Data Collection is created for every Vendor registered with the ARD system but all members reside in a common Channel, which is a private blockchain overlay, where all transacting peers (in this case, the Vendor and Authority peers) must be authorized to perform said transactions. The Vendors receive proper authentication to operate on the Channel when they register with the ARD system. This design drastically reduces the administrative overhead compared to creating new Channels for every Vendor while still catering to the trust and transparency requirements of Responsible Disclosure.

The architecture is comprised of a single chaincode, the `vulnerability-records-contract`. This chaincode is responsible for the creation of the vulnerability records with the corresponding key (in this case, secret s). The CRUD (Create, Read, Update, Delete) operations of the vulnerability records are also handled by this chaincode along with a verify operation that checks if the Channel state hash matches the hash calculated from the vulnerability object to be verified. This function, called the *verifyVulnerability* function, is particularly useful in case of controversy and could be invoked by any Channel member to verify the existence of a transaction. Other Vendors, besides the Vendor responsible for the vulnerability, can see a hash-encrypted copy of the transactions, which could be used for validation and audit purposes, but bear no knowledge of the private states. The hash essentially serves as evidence for the existence of a vulnerability in a particular Vendor's product, but does not allow any unauthorized member to access the private states.

Finally, this chaincode implements the Interledger interfaces for Fabric to receive the secret from and transfer the vulnerability location and related disclosure information to the Public Ledger via the Interledger component. If the secret, which serves as a key to a collection in the private ledger, reveals a valid vulnerability stored in a Private Data Collection, the corresponding vulnerability information is obtained from the Collection. The chaincode then emits an event to

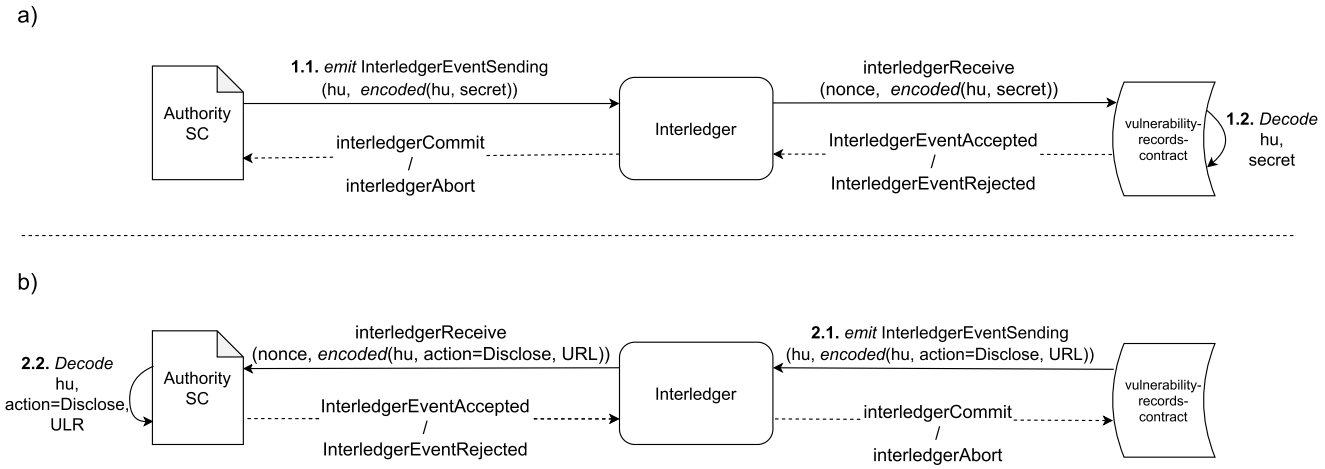


FIGURE 5. A detailed view of the Interledger operations during the disclosure: a) communication from public to private ledger; b) communication from private to public ledgers.

notify Interledger of the acceptance (or rejection if the secret is deemed invalid). If the secret is correct, the chaincode updates the vulnerability information as a document to a Cloudant Database, generates the URL of the web page that displays the corresponding vulnerability details, packs this URL with the corresponding vulnerability hash that identifies the vulnerability, and emits the payload as an event to the Interledger component.

C. INTERLEDGER OPERATIONS

The Interledger component automatically carries out the operations involving both ledgers. To achieve this, the public `Authority SC` in Ethereum and the private `vulnerability-records-contract` in Hyperledger Fabric implement the interfaces provided by the SOFIE Interledger component [43]. Interledger implements the bridging approach [32], meaning that an entity runs an Interledger node that listens for specific events and triggers corresponding functions. In this prototype, the Authority runs the Interledger node. A detailed description of the Interledger component can be found in the paper of Wu *et al.* [44].

When a contract on a ledger wishes to trigger an Interledger operation, this contract emits an *interledgerEventSending(id, payload)* event, with the identifier of an application object (in this case: the vulnerability) as *id* and an encoded message as *payload*. Interledger node then invokes the *interledgerReceive(nonce, payload)* function on the receiving ledger, with *nonce* (a value used by the Interledger component to uniquely track each transaction) and the *payload*. The receiving ledger emits either an *interledgerEventAccepted*, or an *interledgerEventRejected*, event to communicate the positive, or negative, outcome of the operation on the ledger, which is communicated back to the originating ledger by invoking either the *interledgerCommit(id)* or *interledgerAbort(id)* function respectively, with *id* being the same *id* parameter of *interledgerEventSending*. Figure 5 shows the Interledger operations during the disclosure step.

The following sections describe the interledger operations during the approval and the disclosure of the vulnerability u identified by h_u.

1) INTERLEDGER OPERATIONS DURING APPROVAL (STEP 2.a)

The approval of a vulnerability begins with the Authority storing the vulnerability u and the secret s on the private ledger. After the tuple (u, s) is successfully stored on the private ledger, the code dedicated to this operation, the `vulnerability-records-contract` chaincode, emits the *InterledgerEventSending*(h_u, *payload*) event, where *payload* is (h_u, *action* = *Approve*) encoded in bytes. The parameter *action* is a numerical identifier required by the *interledgerReceive* function on the public ledger to distinguish between the approval and disclosure processes. The *payload* is encoded by a chaincode contract in plain bytes with *ethereumjs-abi* library [46] to render it compatible with Solidity abi encoding.

Interledger then calls the *InterledgerReceive(nonce, payload)* function on the public ledger, implemented by the `Authority SC`. The smart contract decodes the payload, sets the state of the vulnerability identified by h_u to `Valid`, and generates the timelocks t_a and t_p for that vulnerability. For simplicity, the timelocks t_a and t_p have a fixed size in the prototype but this can easily be modified based on the deployment policy as discussed in Section VIII. Finally, the `Authority SC` signals the success/failure back to the private ledger using either the *interledgerEventAccepted* or *interledgerEventRejected* event.

2) INTERLEDGER OPERATIONS DURING DISCLOSURE (STEPS 3.b, 4.a OR 4.b)

The disclosure of a vulnerability begins with the Authority publishing the secret s on the public ledger. In this phase, Interledger is triggered twice: first to transfer s from the public to the private ledger as shown in Figure 5 (a) and then

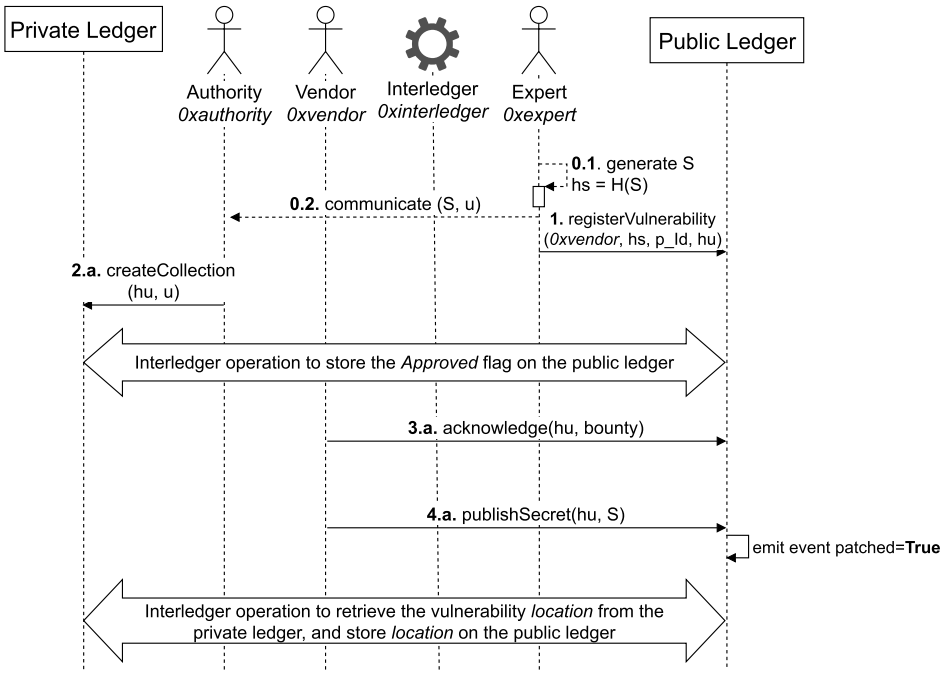


FIGURE 6. Sequence diagram of ARD when the Vendor provides a patch in time. Each function call, represented by a plain arrow, includes the step number as in Figure 1 and the input parameters. A dashed arrow represents an off-ledger operation, for example the Expert generating the hashlock. The Interledger function calls of Figure 5 have been collapsed in two wide arrows.

to transfer the URL from the private to the public ledger as shown in Figure 5 (b).

When called with secret s as input, the *publishSecret* function in the `Authority SC` checks the following conditions:

$$\begin{aligned}&(H(s) = h_s) AND\\ &\quad ((u.State = Valid\ AND\ Timestamp > u.t_a)\ OR\\ &\quad (u.State = Acknowledged\ AND\ Timestamp > u.t_p)\ OR\\ &\quad (u.State = Acknowledged\ AND\ Sender = Vendor))\end{aligned} \tag{1}$$

The first condition implies that the hash of the secret $H(s)$ must match the stored hashlock h_s. The first condition within the OR chain represents the expiration of t_a, the second one the expiration of t_p, and the third one the patch notification from the Vendor. If the boolean expression evaluates to false, the function fails. Otherwise, the public ledger fires the *InterledgerEventSending(h_u, payload)* event to communicate the (h_u, s) as *payload* encoded in bytes[4] to the private ledger (Figure 5 (a)). The state of u is set to `Disclosable`.

In the private ledger, the chaincode is invoked by Interledger calling the *interledgerReceive* function with the received *payload*. The chaincode decodes the payload and verifies the validity of s using the *vulnerabilityExists* function that checks if the vulnerability record matching the specified secret s exists in the Collection. If the function returns `false`, the *InterledgerEventRejected* event is emitted to inform Interledger that no vulnerability exists for the given the secret s, i.e., the secret is invalid.[5] If the function returns `true`, the following steps are carried out: (i) the vulnerability records for s are obtained from the Private Data Collection using the *readVulnerability* function, (ii) the JSON encoded information obtained from the Collection is then uploaded to an external Cloudant database as a new document with the corresponding document identified by h_u, and the URL where the details of this particular vulnerability shall be displayed is generated, and (iii) the *InterledgerEventAccepted* event is emitted.

The private ledger then has to communicate the URL to the public ledger to finalize the disclosure. The chaincode encodes *(h_u, action = Disclose, URL)* as *payload*, with the *action* that identifies the disclosure phase, and the *URL*, where the vulnerability details are published. Finally, the *emitData* function is called to transfer the aforementioned *payload* to the public ledger via Interledger. As illustrated in Figure 5 (b), the private ledger emits the *interledgerEventSending(h_u, payload)*. Interledger catches the event and forwards the *payload* to the public ledger. The *interledgerReceive* function on the public ledger decodes the *payload* to retrieve the URL,[6] stores it, and sets the vulnerability state to `Disclosed`, thereby ending the disclosure process. After the process is completed and the vulnerability

[4]bytes payload = abi.encode(h_u, s) in Solidity.

[5]This actually suggests that the data in the private ledger has been tampered with since the `Authority SC` approved the secret earlier in the process.

[6]hu, action, url = abi.decode(payload, (bytes32, uint, string)) in Solidity.

TABLE 3. Cost in gas and Euro of the smart contract operations. Length of the test URL: 38 characters.

Caller	Smart contract	Function	Action	Step	Cost in gas	Euro
Authority	Authority SC	*deploy*	Deploy an Authority SC	*init*	3,171,402	509.61
Authority	"	*registerVendor*	Register a vendor	*init*	2,837,314	456.89
Expert	"	*registerVulnerability*	Report a vulnerability	1	146,495	23.43
Interledger	"	*interledgerReceive*	Approve a vulnerability	2.a	61,829	9.95
Exp/Aut	"	*publishSecret*	Publish a secret (before acknowledgement)	3.b	110,757	17.5
Vendor	"	*publishSecret*	Publish a patch	4.a	126,686	20.01
Exp/Aut	"	*publishSecret*	Publish a secret	4.b	128,730	20.34
Interledger	"	*interledgerReceive*	Store the URL	3.b, 4.a or 4.b	120,803	19.45
Vendor	Vendor SC	*registerProduct*	Register a product	init	113,311	18.25
Vendor	"	*acknowledge*	Acknowledge a vulnerability	3.a	81,348	13.10

is disclosed, the Authority or the Vendor can use the *deleteVulnerability* function to delete the vulnerability records in the Collection and free up space on the private ledger.

The full protocol involving the case the Vendor provides a patch, with the function invocations, is illustrated as a sequence diagram in Figure 6. The dashed arrows represent the off-ledger operations, the plain arrows represent the on-ledger operations, i.e. transactions, and the wide arrows are a simplification of the Interledger function calls as depicted in Figure 5.

VI. QUANTITATIVE EVALUATION

This section presents the quantitative evaluations of the ARD system using two key metrics: cost and performance.

- *Costs* for all participants, i.e. the money spent on the distributed ledger operations over the whole ARD lifecycle for all parties, including deployment and different processing. The costs should be low enough not to be a disincentive for experts and vendors.
- *Performance* of an ARD system covers the latency of operations, the overall throughput, i.e. number of disclosure transactions per unit of time, and the usage of other computation resources.

Section VI-A provides the gas and Euro costs of the operations performed on the Ethereum ledger for all the participants of the system. Moreover, it evaluates whether the Ethereum throughput is high enough to process the vulnerabilities submitted each year. Section VI-B then provides the benchmark tests of the Hyperledger Fabric setup.

This Section answers the research question 3) *"How does the solution affect the latency and costs associated with the vulnerability disclosure process?"* applied to an implementation with Ethereum, Hyperledger Fabric, and Interledger.

A. COSTS AND THROUGHPUT IN THE PUBLIC LEDGER

Storing information on the Ethereum ledger requires transactions and a corresponding fee in the Ethereum cryptocurrency proportional to the amount of *gas* spent to execute the transactions [28]. The transaction fee is computed by multiplying the amount of gas spent by the *gas price*, i.e. the amount in cryptocurrency (typically in GWei[7]) the account sending the transaction is willing to pay for each unit of gas. Normally, miners favor transactions with higher gas prices, so promising a higher price usually expedites the completion of the transaction, but also raises the cost. Also, each block of Ethereum has a *block gas limit*, i.e. the maximum amount of gas all transactions in a block can spend together.

Table 3 shows the gas cost for the main operations of the ARD smart contracts along with the step numbers used in Section IV: the steps of the initialization, such as the creation of the smart contracts, have been labeled as *init*. The table shows also the cost in Euro of each operation, computed with the following parameters on 12th February 2021: the price of one Ether is 1,464.40€, and the average gas price of the transactions is 110 GWei [47].

The *deploy*ment of a smart contract (`Authority SC` or `Vendor SC`) is the most expensive operation in terms of gas because it needs to deploy the smart contract's bytecode, which is a relatively large amount of data, on Ethereum. However, this is only a one-time cost and amortized over all the thousands of vulnerabilities that could be managed with the system each year it becomes negligible. In contrast, products and vulnerabilities are not stored as smart contracts, and, therefore, their creation costs are cheaper.

Reporting a vulnerability costs ca. 23€ to the Expert, which is a relatively high cost. The operation does not perform complex computations, but it requires the instantiation of a vulnerability record that includes a high number of fields (see Table 2): a trade-off between cost and transparency (criterion 5) can be met by reducing, or increasing, the number of public fields of a vulnerability, or the size of the disclosure data.

The overall cost to report and disclose a vulnerability is ca. 85€. When the Expert discloses a vulnerability, the cost for each participant would be ca. 43 for the Expert, 13€ for the Vendor, and 29€ for Interledger. If the Vendor discloses a vulnerability, the cost would be ca. 23€ for the Expert, 33€ for the Vendor, and 29€ for Interledger. Such costs can be somewhat lowered by the participants by reducing the gas price of each transaction, but if the gas price is too low it may risk being mined very late or even never, which would prevent the disclosure process from finishing (steps 3.b, 4.a, or 4.b).

While the cost for the Expert is relatively high, it is still negligible compared to rewards in vulnerability bounties [48], therefore it is a relatively small price to pay for the Expert to maintain anonymity towards the vendor. As an upside, the

[7] 1 ether (ETH) = 1,000,000,000 GWei

high cost for the Expert can also discourage submissions of fake vulnerabilities from malicious Experts, thus reducing the amount of work for the Authorities.

Assuming a massive submission of vulnerabilities over a short period, the throughput of the smart contract to process them is dependent on the underlying ledger processing capacity. At the time of writing, Ethereum produces blocks roughly every 13 seconds [49], and each block has a gas limit of 12M units [50]. Therefore, an empty block can store at most $12M/146K = 72$ new vulnerability reports, meaning a maximum throughput of about 418,000 reports per day. However, every block is shared among all the participants of the Ethereum blockchain, so the actual reports per day would be much lower due to other traffic.

According to the CVE details dataset [51], 16,556 vulnerabilities were disclosed in 2018, while in 2019 there were 12,174 disclosures. Similarly, in a report published by the European Union Agency for Cybersecurity [52], during the period from January 1st 2018 to August 31st 2019 (608 days) the authors collected a dataset of about 27,000 vulnerabilities from different sources. Assuming that all of these vulnerabilities are communicated through an ARD system uniformly, the system should collect about $16,556/365 = 45.3$, $12,174/365 = 33.35$, or $27,000/608 = 44.4$ vulnerabilities a day on average. Each vulnerability requires between 2-6 Ethereum transactions, potentially spread over months, to be processed. Assuming a relatively even rate of disclosures, ARD requires between 90-272[8] transactions per day, which is a tiny fraction of the potential throughput of Ethereum.

The measurements presented in this section show that: *i)* gas-wise the operations of the system to manage vulnerabilities are not expensive, but Euro-wise the actual cost is not insignificant, especially for the Experts, because they are vulnerable to the fluctuation of both the Ether cryptocurrency and the average gas price. However, compared to the costs of patching the vulnerabilities or the potential costs of unpatched vulnerabilities could incur, the cost of running the ARD system is negligible. *ii)* throughput-wise the system requires only a small portion of the daily transactions on Ethereum, which are over 1 million [53] and the ARD system could, therefore, easily be deployed in the real world.

B. LATENCY, THROUGHPUT, AND RESOURCE UTILIZATION IN THE PRIVATE LEDGER

For the private ledger, the transaction latency, resource utilization, and throughput in Transactions per Second (TPS) were measured to demonstrate that the public ledger is indeed the bottleneck of the whole system.

The private blockchain was benchmarked using Hyperledger Caliper [54], a popular benchmarking tool that measures the performance of a Blockchain application with a set of predefined use cases. Within the network are two Organization types, namely the Authority and the Vendor, with a single Authority peer and 4 Vendor peers. For each benchmark round, 2 test clients were used to generate the test load, with 10 vulnerability records per round. In the Fabric Smart Contracts, CouchDB was used as the state database to store the public and private states. Within the test callback file, all the functionality was tested, namely, creating, reading, updating, and deleting a vulnerability, along with the operations involving information transfer with Interledger. All experiments were performed on a local machine running on Ubuntu 18.04 Operating System, with 4.15.0 Kernel version and Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz. The results are listed in Table 4.

[8] $2 * 45.3$, and $6 * 45.3$

TABLE 4. Chaincode benchmark results using Hyperledger Caliper.

Send Rate (TPS)	Max Latency(s)	Min Latency(s)	Avg Latency(s)	Throughput (TPS)
74.6	0.38	0.01	0.05	74.6

The Caliper measurements confirm that the private ledger built on Hyperledger Fabric framework has negligible contribution to the overall latency of the architecture and that it supports a significantly higher TPS than the public ledger. It is also worth noting that all the operations, including the functions that interact with the Interledger component, are defined within the same chaincode, thereby reducing the number of chaincode installations, approvals, and invocations to one per approved vulnerability. Further, once a Private Data Collection is defined between a Vendor and Authority, the same collection definition could be used to store any further vulnerabilities reported in the same/different products that belong to the same Vendor, thus reducing the administrative overhead incurred by the Authority in setting up Collections. Besides the security aspect of using Private Data Collections to handle the storage [55]–[57], verification, query, and disclosure of vulnerability records, the Fabric Network design also reduces the latency and management concerns by a great extent.

VII. QUALITATIVE ANALYSIS

This section analyses the design of the ARD system, describing in Section VII-A how well the design satisfies the evaluation criteria listed in Section III-B, and in Section VII-B how well the design can resist a vector of potential threats. These together can be summarized as the metric of *Correctness*, which means the vulnerability disclosure is conducted as intended, i.e. securely and reliably while remaining resistant to security risks for each of the participants.

A. QUALITATIVE ASSESSMENT OF THE CRITERIA

The assessment of the evaluation criteria, listed in Section III-B, answers to the first part of research question 1) "*What are the trade-offs between transparency, information confidentiality, and privacy?*".

Privacy (the Expert is protected against intimidation): in the ARD design, the Expert communicates the tuple (u, s) through a secure connection off-ledger, and thereafter only through transactions on the public ledger. The only identifier required from the Expert is on the public ledger. For example, in Ethereum the Expert has an associated pseudonymous

address, i.e., the address is associated with a unique identity. As long as users themselves do not reveal their identity behind the address, their identity is considered protected with a high level of confidence. Non-reusing the same address, like in Bitcoin, can help maintain this high level of privacy [58]. However, receiving and spending the bounty may put the Expert's privacy at risk with clustering and de-anonymization techniques that can be processed on a public ledger [59], [60]. To mitigate this risk and improve their privacy, Experts can rely on mixing services to conceal their public spending traces [61]. Overall, the ARD system protects the Expert's privacy well.

Early disclosure (if the Vendor does not acknowledge the vulnerability, the Expert is allowed to immediately disclose the vulnerability): a condition to disclose a vulnerability (see Condition 1, Section V-C) is the expiration of the first grace period t_a, i.e. the time the Vendor has to acknowledge the vulnerability. If the Vendor does not respond to the acknowledgment request, the smart contract allows the Authority or the Expert to trigger the Disclose procedure by publishing the secret.

Automation (the disclosure approach should be as automatic as possible, requiring only a minimal amount of manual intervention): the automation of the disclosure of a vulnerability, either after the grace period or with a patch-release, is achieved through the Interledger component. After acquiring the secret from a smart contract function on the public ledger, interledger is able to read the vulnerability data (or its location) from the private ledger and store it on the public ledger. However, currently, smart contracts cannot normally self-activate, therefore the expiration of the grace period cannot be automatically detected by the contracts alone. This could be solved with tools, such as Ethereum Alarm Clock [62], which schedule transactions in the future.

Secrecy until disclosure (the secrecy of the acknowledged vulnerability is preserved until disclosure): the secrecy of the sensitive vulnerability details is ensured by the private ledger. How this is accomplished depends on the private ledger capabilities. The prototype demonstrates an approach using the chaincode logic and collection policies provided by the Hyperledger Fabric framework. For every accepted vulnerability report, the Authority creates a record in the Private Data Collections and the corresponding collection definition only grants read/write access to the Authority and the Vendor whose product is affected. The Interledger component can only access the information that is sent by collection using events. While a single Channel is shared across all Vendors (and the Authority), the other Vendors do not have access to the Private States. Instead, they see a hash-encrypted copy of the transactions in the Collection, which can be used for validation and audit purposes. This approach increases the transparency of transactions within the consortium while avoiding the administrative overhead and latency concerns that would be incurred if the actors were to create a separate Channel per Vendor. Further, the private details stored in these Collections are also not revealed to the Ordering Service (comprising of nodes with the responsibility to arrange batches of submitted transactions into a well-defined sequence and package them into blocks for the Hyperledger Fabric network), thereby adding another layer of security.

Transparency (the information about the whole process of vulnerability disclosure is available to the General Public): the operations computed by the smart contracts in the public ledger provide the ARD system the transparency of the process. The general public is aware of the existence of the vulnerability u, along with the additional information stored in the metadata m_u, and is also aware of the patching process by virtue of the states illustrated in Figure 4. Also, the information on the acknowledgment and the bounty offered by the Vendor is also made public, which serves as a display of the Vendor's willingness and dedication towards fixing vulnerabilities and improving the quality of their products. Moreover, a transparent system pushes all parties to adhere to the rules: it allows the general public to validate the operations by recomputing them. Combined with the resistance to modifications and the non-repudiation, if the Authority does not accept a valid vulnerability, then the Expert can reveal it and anyone can check that the hash of the revealed vulnerability matches the hash of the rejected one stored on the public ledger, indicating either foul behavior by the Authority or an erroneous evaluation of the validity of the vulnerability submission.

Therefore, an ARD system can successfully satisfy all the evaluation criteria of the *Correctness* metric proving it to be better than the previous disclosure solutions.

B. POTENTIAL THREATS

This section analyses the potential attacks that any party can mount against the ARD system using a threat model similar to Satija *et al.* [63].

1) GENERAL PUBLIC

An adversary belonging to the general public can be anyone, including a Vendor, state actor, or a malicious hacker. A motivation driving an attack could be to gain knowledge of the vulnerability to exploit it, to use it for industrial espionage, or to sell it in the black market. In the ARD, the secrecy of the vulnerability is protected by the private ledger, and the attacker would need the same permission level as the Authority, or breach the private ledger, to be able to access a vulnerability.

Another motivation for the attacker would be the collection of bounties. Here, the attacker would need to modify the transactions on the public ledger, an action that is prevented by the ledger itself on account of being immutable with a very high level of confidence. Any other modification of the smart contracts is prevented in the same way.

An adversary may also flood the system with bogus vulnerability reports, trying the undermine the reputation of the system or a Vendor, since the Vendor's reputation may be affected by the number of open issues in their products. However, this attack is discouraged by the fees of public

ledgers, and prevented by the Authority, since their task is to filter bogus reports, and in the ARD system, a vulnerability is not considered valid until it is approved.

2) EXPERT

A malicious Expert may wish to access previously reported vulnerabilities. However, even if the Expert reports a vulnerability, the Expert is not allowed to participate in the private ledger consortium nor access any of the vulnerability information (including the one they themselves submitted).

An Expert may also try to disclose the vulnerability before the grace period has expired and still collect the bounty. However, the smart contracts protect the bounty from early disclosures (unless the grace period t_a has elapsed). Moreover, the attempt will be recorded in the public ledger, even if unsuccessful. Since the ARD system cannot prevent a malicious Expert from publishing the vulnerability outside the system, the only action that can be taken is the revocation of the bounty. This can be achieved through a smart contract function that is invocable only by the Authority in case an unauthorized disclosure outside of the system is detected and deemed to likely be the fault of the Expert. Thus, it's not in the interest of the Expert to leak the vulnerability on any other forum before the grace period has expired as that would risk their ability to collect the bounty.

3) VENDOR

A Vendor registered to the ARD might act maliciously to avoid the costs of fixing a vulnerability or paying the bounties. In the former case, a Vendor may not acknowledge any vulnerabilities, but this allows the Expert to disclose the vulnerability earlier (after the acknowledgment grace period t_a has elapsed): owing to the transparency of the system, a customer can see that a Vendor never agrees with Experts and Authorities to fix issues in their products, so the Vendor risks losing the trust of their customers. In the latter case, a Vendor may never pay the bounties to Experts but everyone can view this information (due to the transparency of public ledgers) and lose trust in that Vendor.

Alternatively, a Vendor may attempt to avoid paying for the bounties by disclosing a vulnerability outside the ARD system while pretending to be the Expert. However, this is not a reasonable solution for the Vendor, for two reasons: (i) disclosing the vulnerability too early might have worse consequences for the Vendor than paying the bounty, and (ii) the Vendor knows the Expert's public ledger address only and is therefore not able to provide authentication for the disclosure, i.e. a valid digital signature matching the Expert's public key.

The Vendor may also employ malicious techniques such as value-fingerprinting attacks to de-anonymize the Expert [64] to e.g. intimidate them. To protect from attacks that exploit address reuse (a practice prevalent in account-based blockchains such as Ethereum), trustless mixing techniques [61] could be employed in the public ledger design.

Finally, since a Vendor participates in the private ledger, they may try to access the vulnerability information of their competitors. However, this can be prevented if the private ledger provides a suitable data access control mechanism, like Private Data Collections in Hyperledger Fabric. As demonstrated by the prototype, a Collection is shared only between the Authority and the Vendor in question, and no other member on the Channel is privy to the sensitive vulnerability records stored in the Collection. The Channel members can only see a hash of the Private states, which could be used for verification and audit purposes. Further, since proof of every transaction is made available to all members on the Channel, if the Vendor exhibits questionable behavior (such as deleting a vulnerability before it is disclosed), the Authority is made aware of such malpractice and can take the requisite steps against the Vendor.

4) AUTHORITY

A potential threat is the difficulty to know whether the Authority's decision to approve or reject a vulnerability submission is justified. If the Authority rejects a valid vulnerability, they are likely to lose their reputation, since the Expert can always release the vulnerability details through other means.

Since the Authority is managing the website for vulnerability submissions, it can theoretically employ browser-based fingerprinting attacks to try to de-anonymize the Expert. However, the likelihood of a successful attack is low, since Expert does not submit vulnerabilities frequently, and the Expert can easily use combinations of various means such as using a separate browser profile, Tor Browser, VPN, proxy, etc. to defend from the fingerprinting during the submission of the vulnerability [65].

It is important to note that the Authority might serve as the Governing body in the private ledger. For example, in the prototype, the Authority is responsible for making decisions on membership criteria and access rights for all nodes requesting to join a Channel within the Fabric network. Moreover, in the prototype implementation, the Authority runs the Interledger component as well. This allows the malicious Authority (or the attacker that has breached the Authority) to prevent automatic disclosure of the vulnerability after it has been approved. Redundant Interledger nodes could reduce the impact of such attack.

Overall, the Authority plays a central role in the ARD system and, if working alone, is the main trusted party of the system. The power and importance of a single Authority can, however, be mitigated, if they belong to a consortium of Authorities jointly running the ARD system. Such a distributed trust has successfully been utilized e.g. in the DLTs themselves to reduce the level of trust required on this key intermediary.

VIII. DISCUSSION

The ARD system solves the key problems of responsible disclosure, such as the lack of transparency and privacy concerns for the Experts, while automating much of the disclosure

process. Other solutions involving DLTs, such as the one from Hoffman *et al.* [40] and the Sentinel Protocol [39], do not entirely solve these problems, as described in Section III-A, though, e.g. the Sentinel Protocol could be integrated into the ARD to further reduce the reliance on the Authority in the Approval step by delegating the validation of the vulnerability submission to a separate consortium of security experts. The deployment of the ARD solution is also quite feasible because it does not require a complex setup or high costs.

Assuming a society with an extensive deployment of the ARD systems, the existence of the vulnerabilities, their evaluations, and the associated bounties would be publicly recorded with no possibility of repudiation. This should motivate honest behavior from the Experts, Vendors, and Authorities because false statements can neither be removed nor hidden. As a consequence, the General Public becomes more informed about security vulnerabilities and can better understand their severity from how quickly the Vendors promise to fix them and how much they reward the Experts. In the long run, this should raise the awareness of the General Public about the digital world, its advantages, and especially its risks.

As analyzed in Section VII-A, the ARD design protects the Expert's identity, since the Expert uses a pseudonymous identity when submitting the vulnerability information on the ledger. However, there is also no additional information about the person who reports a vulnerability, which could have been helpful to understand their expertise in cybersecurity. To fill this gap, the design could integrate *Verifiable Credential* (VC) and *Decentralized Identifier* (DID) technologies [66], [67] to allow the Experts to (anonymously) demonstrate their past contributions. As a result, submissions from verified Experts can increase the awareness of the issue, and push Vendors to fix the vulnerability faster. Moreover, since evaluating submitted vulnerabilities might be a long process, extra information on who submitted the reports might help the decision to ignore potentially bogus submissions.

To answer the research question 2) "*How does automation of the process affect the disclosure of vulnerabilities?*", implementing the key steps of the disclosure process with interledger and DLTs to automate the important decisions of the process can prevent manual decisions from being too flexible. First, generating the timelocks as a function on the public ledger instead of being inputs from the Authorities avoids exceptions to the rules to e.g. favor certain Vendors or penalize others. Further, the interledger operations that facilitate the disclosure of a vulnerability are triggered with a single manual operation, i.e. revealing the secret, thus reducing the number of potential human errors. A further benefit of the choice of using two ledgers and automation is that it can also support extraordinary circumstances preventing the disclosure of the vulnerability, for instance after a court order: if for any reason the vulnerability data cannot be disclosed, a related action can be fired on the private ledger, and a special state, indicating the vulnerability will currently not be disclosed, can be added to the public ledger that will be automatically set with an interledger operation. The required information, e.g. reference to the relevant court order, should then be published, instead, to protect the Vendor from being accused of trying to hide the vulnerability.

Though the prototype chose to implement automated timelock generation with fixed time periods, fundamentally the choice of grace periods is not an ARD architecture choice but a deployment policy decision - and the ARD implementation can then be modified, if necessary, to support the new policy. For instance, it would be possible to allow the Authority to set the time period after negotiation with the Vendor or it could be based on the recommendation of the security experts in the Sentinel Protocol. Alternatively, given the difficulty of always choosing the best grace periods, it would be possible to allow the Vendor to request an extension to the timelock in case the vulnerability turns out to be particularly difficult to fix. Similarly, it would be possible to provide support for re-opening supposedly fixed vulnerabilities if the provided patch turns out to be insufficient. Implementing any of the above changes would require only minor changes to the ARD implementation, but their introduction needs to be weighed against the potential misuses the added flexibility enables. However, in all cases, any favoritism/penalizing or slow fixing would then be apparent from the public ledger. After all, the main idea of the ARD system is to improve security through transparency, and therefore Vendors that produce insecure products and fail to fix the vulnerabilities quickly should be penalized through the loss of reputation while Vendors doing the opposite should gain reputation. To that end, regardless of the method of how the timelock period was chosen, it should be short enough in order to provide real incentives for Vendors to fix the vulnerabilities quickly.

In ARD, the Authority plays a central role in arbitrating between the interests of individual Vendors and the public and in weeding out false vulnerability reports. This enables many of the benefits of ARD, but it also requires trusting the Authority to behave properly. Ideally, even this step could be automated, but currently such solutions are not technically feasible, so the next best solution to reduce the required trust is to have multiple Authorities working as a consortium, which helps prevent any single Authority from misbehaving. Also, utilizing a separate consortium of experts to evaluate the submissions as suggested by the Sentinel Protocol would further reduce the trust requirement on the Authority, but would then also introduce a new party that needs to be trusted with the sensitive vulnerability details.

In addition, the combination of a reward system and VC and DID technologies would allow Experts to take credit for their contributions as the Expert could at any time connect its ledger identity either to a pseudonym, or to their real identity, and could help Experts and Vendors to choose a trustworthy Authority in a case there are multiple competing Authorities offering ARD services. Also, a reputation mechanism could be added for the Vendors and the Authorities to further motivate their actions [68]–[71].

Finally, because it is built with DLTs and interledger technologies, the ARD system also inherits some of their flaws as well [32]–[34], [55]–[57], [72]–[75]. For instance, the high fees on the public ledger can discourage Experts from reporting the vulnerabilities since it may be too costly for them, especially when they do not seek rewards or if their reports are rejected by the Authorities. However, since the transparency and immutability of an ARD system depend on the public ledger's security, a high price of the cryptocurrency is synonymous with higher participation, and consequently, a greater level of security of the ledger. Similarly, the more popular a private ledger framework is, the more effort is likely put into further improving its security. By combining multiple ledgers, interledger technologies utilize the strengths of different DLTs while avoiding many of the drawbacks. Ultimately, it's up to the deployers to carefully consider all of these aspects, whose combination will determine the balance between transparency, security, privacy, and decentralization of the ARD system.

IX. CONCLUSION

This paper presents a solution to improve the current state of the art concerning the disclosure of vulnerabilities, based on the idea of Automated Responsible Disclosure (ARD). A complete and detailed design of the solution is presented, together with an open-source implementation that leverages Ethereum, Hyperledger Fabric, and the SOFIE Interledger component. The paper benchmarks the solution on several performance parameters, evaluates its resistance to a set of attack vectors, and addresses the research questions about the different trade-offs in system design.

In the currently used Responsible Disclosure approach, the Experts reporting the vulnerabilities suffer from a lack of privacy as they are often required to identify themselves, thereby risking intimidation and legal recourse. Also, the lack of transparent communication fosters a false sense of security among the general public. Finally, the approach still allows Vendors to arbitrarily delay the disclosure of a vulnerability.

The ARD solution presented in this paper tackles all the aforementioned problems. The combination of a public and a private ledger achieves both a transparent vulnerability management scheme, and secrecy-oriented storage of the vulnerability data. In an ARD system, the Expert is not required to identify themselves but uses an address in the public ledger instead, which improves privacy and avoids threats from Vendors. In addition, the interledger functionalities automate the key steps of the entire vulnerability disclosure process, bridging the communication between the two ledgers. Finally, the transactions on the public ledger allow anyone to find out about the existence of a vulnerability in a digital system, and the current stage of its resolution. As shown by the prototype implementation, the costs of the transactions are negligible compared to the costs of the vulnerabilities themselves, and the amount of transactions required by the ARD system is low.

REFERENCES

[1] M. Lezzi, M. Lazoi, and A. Corallo, "Cybersecurity for industry 4.0 in the current literature: A reference framework," *Comput. Ind.*, vol. 103, pp. 97–110, Dec. 2018.

[2] Bruce Schneier. *A Plea for Simplicity You Can't Secure What You Don't Understand*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.schneier.com/essays/archives/1999/11/a_plea_for_simplicit.html

[3] Threat Post. *2 Million IoT Devices Vulnerable to Complete Takeover*. Accessed: Mar. 30, 2021. [Online]. Available: https://threatpost.com/iot-devices-vulnerable-takeover/144167/

[4] L. J. Trautman and P. C. Ormerod, "Industrial cyber vulnerabilities: Lessons from stuxnet and the Internet of Things," *Univ. Miami Law Rev.*, vol. 72, no. 3, p. 761, 2017.

[5] A. Nakajima, T. Watanabe, E. Shioji, M. Akiyama, and M. Woo, "A pilot study on consumer IoT device vulnerability disclosure and patch release in Japan and the United States," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Jul. 2019, pp. 485–492.

[6] *Public Bug Bounty Program List*. [Online]. Available: https://www.bugcrowd.com/bug-bounty-list/

[7] ENISA. *Economics of Vulnerability Disclosure*. [Online]. Available: https://www.enisa.europa.eu/publications/economics-of-vulnerability-disclosure

[8] M. McQueen, J. L. Wright, and L. Wellman, "Are vulnerability disclosure deadlines justified?" in *Proc. 3rd Int. Workshop Secur. Meas. Metrics*, Sep. 2011, pp. 96–101.

[9] W. Pond. *Do Security Holes Demand Full Disclosure?* Accessed: Mar. 30, 2021. [Online]. Available: https://www.zdnet.com/article/do-security-holes-demand-full-disclosure/

[10] A. Stone, "Software flaws, to tell or not to tell?" *IEEE Softw.*, vol. 20, no. 1, pp. 70–73, Jan. 2003.

[11] *The Vulnerability Disclosure Process: Still Broken*. Accessed: Mar. 30, 2021. [Online]. Available: https://threatpost.com/the-vulnerability-disclosure-process-still-broken/137180/

[12] *A Bizarre Twist in the Debate Over Vulnerability Disclosures*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.wired.com/2015/09/fireeye-enrw-injunction-bizarre-twist-in-the-debate-over-vulnerability-disclosures/

[13] *Xerox Legal Threat Reportedly Silences Researcher at Infiltrate Security Conference*. Accessed: Mar. 30, 2021. [Online]. Available: https://portswigger.net/daily-swig/xerox-legal-threat-reportedly-silences-researcher-at-infiltrate-security-conference

[14] *Boeing's Poor Information Security Posture Threatens Passenger Safety, National Security, Researcher Says*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.csoonline.com/article/3451585/boeings-poor-information-security-posture-threatens-passenger-safety-national-security-researcher-s.html

[15] Github. *Research Threats: Legal Threats Against Security Researchers*. Accessed: Mar. 30, 2021. [Online]. Available: https://github.com/disclose/research-threats

[16] D. Lagutin, Y. Kortesniemi, V. Siris, N. Fotiu, G. Polyzos, and L. Wu, *Leveraging Interledger Technologies in IoT Security Risk Management*. New York, NY, USA: Now, 2020, pp. 228–246.

[17] CVE. *Terminology*. Accessed: Mar. 30, 2021. [Online]. Available: https://cve.mitre.org/about/terminology.html

[18] Enisa. *The EU Cybersecurity Agency, and Repealing Regulation (EU) 526/2013, and on Information and Communication Technology Cybersecurity Certification ('Cybersecurity Act')*. Accessed: Mar. 30, 2021. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2019/881/oj

[19] *NIS Directive*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.enisa.europa.eu/topics/nis-directive

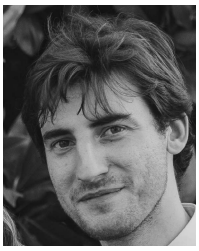
[20] International Organization for Standardization. *Information Technology—Security Techniques—Vulnerability Disclosure*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.iso.org/standard/72311.html

[21] *Information Technology—Security Techniques—Vulnerability Handling Processes*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.iso.org/standard/69725.html

[22] National Cyber Security Centre. *Coordinated Vulnerability Disclosure: The Guideline*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.enisa.europa.eu/news/member-states/WEB_115207_BrochureNCSC_EN_A4.pdf

[23] U. Ķinis, "From responsible disclosure policy (RDP) towards state regulated responsible vulnerability disclosure procedure (hereinafter–RVDP): The Latvian approach," *Comput. Law Secur. Rev.*, vol. 34, no. 3, pp. 508–522, 2018.

[24] NPR. *Uber Pays 148 Million Dollar Over Yearlong Cover-up of Data Breach*. [Online]. Available: https://www.npr.org/2018/09/27/652119109/uber-pays-148-million-over-year-long-cover-up-of-data-breach?t=16023
[25] S. Ølnes, J. Ubacht, and M. Janssen, "Blockchain in government: Benefits and implications of distributed ledger technology for information sharing," *Government Inf. Quart.*, vol. 34, no. 3, pp. 355–364, 2017.
[26] K. Wust and A. Gervais, "Do you need a blockchain?" in *Proc. Crypto Valley Conf. Blockchain Technol. (CVCBT)*, Jun. 2018, pp. 45–54.
[27] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Mar. 30, 2021. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[28] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2014.
[29] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, and D. Enyeart, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.
[30] *Hyperledger Indy, Readthedocs*. Accessed: Mar. 30, 2021. [Online]. Available: https://indy.readthedocs.io/en/latest/
[31] Ethereum Documentation. *On Sharding Blockchains*. Accessed: Mar. 30, 2021. [Online]. Available: https://eth.wiki/sharding/Sharding-FAQs
[32] V. A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, and G. C. Polyzos, "Interledger approaches," *IEEE Access*, vol. 7, pp. 89948–89966, 2019.
[33] T. Koens and E. Poll, "Assessing interoperability solutions for distributed ledgers," *Pervas. Mobile Comput.*, vol. 59, Oct. 2019, Art. no. 101079.
[34] V. Buterin. *Chain Interoperability*. Accessed: Mar. 30, 2021. [Online]. Available: https://allquantor.at/blockchainbib/pdf/buterin2016chain.pdf
[35] M. Herlihy, "Atomic cross-chain swaps," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2018, pp. 245–254.
[36] S. Ransbotham and S. Mitra, "The impact of immediate disclosure on attack diffusion and volume," in *Economics of Information Security and Privacy III*, B. Schneier, Ed. New York, NY, USA: Springer, 2013, doi: 10.1007/978-1-4614-1981-5_1.
[37] H. Cavusoglu, H. Cavusoglu, and S. Raghunathan, "Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge," *IEEE Trans. Softw. Eng.*, vol. 33, no. 3, pp. 171–185, Mar. 2007.
[38] A. Arora, R. Krishnan, R. Telang, and Y. Yang, "An empirical analysis of software Vendors' patch release behavior: Impact of vulnerability disclosure," *Inf. Syst. Res.*, vol. 21, no. 1, pp. 115–132, Mar. 2010.
[39] Uppsala Security. *Protect Your Cryptocurrencies With Advanced Software Solutions From Uppsala Security*. Accessed: Mar. 30, 2021. [Online]. Available: https://uppsalasecurity.com/whitepapers/sentinelprotocol_whitepaper_english.pdf
[40] A. Hoffman, E. Becerril-Blas, K. Moreno, and Y. Kim, "Decentralized security bounty management on blockchain and IPFS," in *Proc. 10th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2020, pp. 0241–0247.
[41] *Web3js LIBRARY*. Accessed: Mar. 30, 2021. [Online]. Available: https://web3js.readthedocs.io/en/v1.3.0/
[42] *ARD Repository*. Accessed: Mar. 30, 2021. [Online]. Available: https://github.com/SOFIE-project/ARD-Implementation
[43] Github. *Sofie Interledger Repository*. Accessed: Mar. 30, 2021. [Online]. Available: https://github.com/SOFIE-project/Interledger
[44] L. Wu, Y. Kortesniemi, D. Lagutin, and M. Pahlevan, "The flexible interledger bridge design," in *Proc. 3rd Conf. Blockchain Res. Appl. Innov. Netw. Services (BRAINS)*, Sep. 2021, pp. 69–72.
[45] Github. Sofie interledger repository. Accessed: Mar. 30, 2021. [Online]. Available: https://github.com/SOFIE-project/Interledger/blob/master/doc/adapter-eth.md
[46] npmjs. *Ethereumjs-Abi Library*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.npmjs.com/package/ethereumjs-abi
[47] *Etherscan, Gas Tracker*. Accessed: Mar. 30, 2021. [Online]. Available: https://etherscan.io/gastracker
[48] *Github Security Bug Bounty*. Accessed: Mar. 30, 2021. [Online]. Available: https://bounty.github.com/index.html
[49] *Etherscan, Block Time*. Accessed: Mar. 30, 2021. [Online]. Available: https://etherscan.io/chart/blocktime
[50] *Etherscan, Gas Limit*. Accessed: Mar. 30, 2021. [Online]. Available: https://etherscan.io/chart/gaslimit
[51] *CVE Details*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.cvedetails.com/
[52] Enisa. *State of Vulnerabilities 2018/2019–Analysis of Events in the Life of Vulnerabilities*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.enisa.europa.eu/publications/technical-reports-on-cybersecurity-situation-the-state-of-cyber-security-vulnerabilities
[53] *Etherscan, Transactions Per Day*. Accessed: Mar. 30, 2021. [Online]. Available: https://etherscan.io/chart/tx
[54] Github. *Hyperledger Caliper*. Accessed: Mar. 30, 2021. [Online]. Available: https://github.com/hyperledger/caliper
[55] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun., "Potential risks of hyperledger fabric smart contracts," in *Proc. IEEE Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Feb. 2019, pp. 1–10.
[56] S. Hua, S. Zhang, B. Pi, J. Sun, K. Yamashita, and Y. Nomura, "Reasonableness discussion and analysis for hyperledger fabric configuration," 2020, *arXiv:2005.11054*.
[57] N. Andola, Raghav, M. Gogoi, S. Venkatesan, and S. Verma, "Vulnerabilities on hyperledger fabric," *Pervas. Mobile Comput.*, vol. 59, Oct. 2019, Art. no. 101050.
[58] *Bitcoin Wiki: Address Reuse*. Accessed: Mar. 30, 2021. [Online]. Available: https://en.bitcoin.it/wiki/Address_reuse
[59] G. Fanti and P. Viswanath, "Deanonymization in the bitcoin P2P network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1364–1373.
[60] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and Privacy in Social Networks*, Y. Altshuler, Y. Elovici, A. Cremers, N. Aharony, and A. Pentland, Eds. New York, NY, USA: Springer, 2013, doi: 10.1007/978-1-4614-4139-7_10.
[61] J. H. Ziegeldorf, R. Matzutt, M. Henze, F. Grossmann, and K. Wehrle, "Secure and anonymous decentralized bitcoin mixing," *Future Gener. Comput. Syst.*, vol. 80, pp. 448–466, Mar. 2018.
[62] *Ethereum Alarm Clock*. [Online]. Available: https://www.ethereum-alarm-clock.com/
[63] S. Satija, A. Mehra, S. Singanamalla, K. Grover, M. Sivathanu, N. Chandran, D. Gupta, and S. Lokam, "Blockene: A high-throughput blockchain over mobile devices," in *Proc. 14th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2020, pp. 567–582.
[64] F. Béres, I. András Seres, A. A. Benczúr, and M. Quintyne-Collins, "Blockchain is watching you: Profiling and deanonymizing ethereum users," 2020, *arXiv:2005.14051*.
[65] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, "Browser fingerprinting: A survey," *ACM Trans. Web*, vol. 14, no. 2, pp. 1–33, 2020. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3386040
[66] *Verifiable Credentials Data Model 1.0*. Accessed: Mar. 30, 2021. [Online]. Available: https://w3c.github.io/vc-data-model/
[67] *Decentralized Identifiers (DIDs) V1.0*. Accessed: Mar. 30, 2021. [Online]. Available: https://www.w3.org/TR/did-core/
[68] A. Bogliolo, P. Polidori, A. Aldini, W. Moreira, P. Mendes, M. Yildiz, C. Ballester, and J.-M. Seigneur, "Virtual currency and reputation-based cooperation incentives in user-centric networks," in *Proc. 8th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Aug. 2012, pp. 895–900.
[69] F. Hendrikx, K. Bubendorfer, and R. Chard, "Reputation systems: A survey and taxonomy," *J. Parallel Distrib. Comput.*, vol. 75, pp. 184–197, Jan. 2015.
[70] K. N. Khaqqi, J. J. Sikorski, K. Hadinoto, and M. Kraft, "Incorporating seller/buyer reputation-based system in blockchain-enabled emission trading application," *Appl. Energy*, vol. 209, pp. 8–19, Jan. 2018.
[71] A. Lisi, A. De Salve, P. Mori, L. Ricci, and S. Fabrizi, "Rewarding reviews with tokens: An ethereum-based approach," *Future Gener. Comput. Syst.*, vol. 120, pp. 36–54, Jul. 2021.
[72] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (SoK)," in *Principles of Security and Trust* (Lecture Notes in Computer Science), vol. 10204, M. Maffei and M. Ryan, Eds. Berlin, Germany: Springer, 2017, doi: 10.1007/978-3-662-54455-6_8.
[73] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Gener. Comput. Syst.*, vol. 107, pp. 841–853, Jun. 2020.
[74] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–43, 2020.
[75] F. Benhamouda, S. Halevi, and T. Halevi, "Supporting private data on hyperledger fabric with secure multiparty computation," *IBM J. Res. Develop.*, vol. 63, nos. 2–3, p. 3, Mar./May 2019.



ANDREA LISI received the B.Sc. and M.Sc. degrees in computer science from the Department of Computer Science, University of Pisa, Italy, in 2016 and 2019, respectively, where he is currently pursuing the Ph.D. degree in computer science.

His current research interests include in the field of cryptocurrency and blockchain technology, in particular he focuses on layer-2 techniques to reduce the costs of an application while preserving the benefits of blockchains.

PRATEETI MUKHERJEE is currently pursuing the bachelor's degree in computer science and engineering with the Institute of Engineering and Management, India.

She is a Research Assistant at Aalto University, Finland, where she worked on EU H2020 Project SOFIE. Her research interests include blockchain technology, security and cryptography, distributed systems, and the Internet of Things.

LAURA DE SANTIS was born in Salerno, Italy, in 1991. She received the M.S. degree in electronic engineering from the University of Salerno, in 2018, where she is currently pursuing the Ph.D. degree in industrial engineering.

She is collaborating with the Aalto University in the SOFIE project in development of a secure blockchain-based IoT framework. Her research interests include the IoT security and blockchain technology. Her research in this project is focused on interledger protocol for the distributed ladger technology.

LEI WU received the B.S. (Eng.) degree in electronic science and technology from the University of Electronic Science and Technology of China, Chengdu, China, in 2012, and the Ph.D. degree in circuits and systems from Nanyang Technological University, Singapore, in 2018.

He has been working as a Postdoctoral Researcher on EU H2020 projects SOFIE and IoT-NGIN at Aalto University, since 2019. His current research interests include distributed ledgers and blockchain technologies, decentralised architectures and applications, privacy protection, and the Internet of Things.

DMITRIJ LAGUTIN received the M.Sc. (Tech.) degree from the Helsinki University of Technology, Finland, in 2005, and the D.Sc. (Tech.) degree from Aalto University, Finland, in 2010.

He was a Researcher in several research projects with the Helsinki University of Technology and Aalto University, including EU FP7 PSIRP, PURSUIT, and EU H2020 POINT projects. He is currently a Coordinator and a Research Fellow with the EU Horizon 2020 SOFIE Project at Aalto University. His research interests include network security and privacy, the Internet of Things, blockchains, and future network technologies.

YKI KORTESNIEMI received the M.Sc. (Tech.) degree in industrial management and the Lic.Sc. (Tech.) degree in computer science from the Helsinki University of Technology, Finland, in 1998 and 2003, respectively, and the D.Sc. (Tech.) degree in networking technology from Aalto University, Finland, in 2015.

He has worked on numerous research projects at the Helsinki University of Technology and Aalto University, including the EU H2020 projects SOFIE and IoT-NGIN. His research interests include information security and privacy, data protection, MyData and legal design, the Internet of Things, distributed ledgers and blockchains, and decentralized identifiers and verifiable credentials.