Pervasive and Mobile Computing 59 (2019) 101038

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Secure blockchain in the enterprise: A methodology

Cédric Hebert, Francesco Di Cerbo *

SAP Security Research, France

ARTICLE INFO

Article history:
Received 7 December 2018
Received in revised form 10 June 2019
Accepted 20 June 2019
Available online 25 June 2019

Keywords:
Threat modeling
Blockchain
Risk assessment

ABSTRACT

The adoption of blockchain in a software architecture should be perceived as a double-edged sword: if on the one hand, it allows to achieve benefits in terms of auditability, strong integrity protection and virtual tokens management, on the other hand they come together with peculiarities, like immutability, that need to be properly considered; for example, in the interaction with personal data, blockchain adoption may result particularly critical. Therefore, in order to guide software architects wishing to leverage the potential of an existing blockchain technology for their solutions, we propose a list of security threats specific to blockchain, integrated in a multi-staged architecture analysis approach based on threat modeling. It comprises methods for the evaluation and identification of the most suitable blockchain technology (if at all viable) for the scenario, coupled with design analysis processes which consider web application — as well as our blockchain-specific security threats. We illustrate our approach through a paradigmatic example.

1. Introduction

The term "blockchain" has become popular in the last decade, also for the general public. Mainstream media used it extensively, especially to discuss and explain crypto-currencies, and the concepts were studied and analyzed by the finance research community (as examples, see [1–3]); Bitcoin [4] in particular, whose importance is demonstrated by observing its market capitalization, about 5B dollars in April 2019 [5]. Such interest is also motivated by the properties of blockchain technologies and their promises of decentralization, robustness, immutability, transparency and at the same time, anonymity. It is only natural, therefore, that the software engineering community begin to adopt the blockchain, in different technology flavors [6]. However, blockchain adoption requires specific cares, spanning from an understanding of the underpinnings to architectural and implementation details, that are not necessarily adopted everywhere [7]. Recent studies in the international development sector [8] seem to confirm this perception.

In this work, we propose our methodology for a secure adoption of blockchain technologies in enterprise software development projects. It relies on threat modeling as means to achieve a security-by-design and sound approach to software security. The methodology guides software architects in understanding if blockchain adoption should be considered or not, and in case, which flavors to consider. Subsequently, we focus on the security implications of blockchain adoption, by including in our proposal an extension to a well-known and industrial-adopted architecture review methodology, the Microsoft-developed threat modeling approach STRIDE [9]. We do so by proposing a list of blockchain-specific security risks. We aim in this way at maximizing the general applicability of our methodology, considering the broad scope of STRIDE.

To exemplify our methodology, we will follow the activities of a persona character, Alice, a software architect, that is about to start a new software project considering the inclusion of blockchain technologies.

* Corresponding author.
E-mail address: francesco.di.cerbo@sap.com (F. Di Cerbo).

https://doi.org/10.1016/j.pmcj.2019.101038
1574-1192/

> Alice is starting her new software project. And as long as she remembers, she has been a big fan of retrogaming. Back in the times, she was spending countless hours playing shooter games at the arcade, always aiming at reaching place 1 in the leaderboard.
>
> Today she thinks about giving the new generation the thrill of this experience in a new kind of video game.
>
> She has the idea of a coin-operated game and of a global leaderboard; this makes her think that she may use a blockchain solution. A few days of brainstorming later, she comes-up with an initial design. Now is the time to validate the concept from a security point of view by applying our blockchain-specific threat modeling methodology.

Threat modeling can be defined as "an approach for creating an abstraction of a software system, aimed at identifying attackers' abilities and goals, and using that abstraction to generate and catalog possible threats that the system must mitigate". It can integrate the early stages of a software development process, where software architectures are reviewed from a security perspective, effectively materializing a security-by-design approach. Among the threat modeling methods, we chose the Microsoft-developed STRIDE for its industrial adoption and support tool availability.[1] The methodology includes the analysis of a software architecture and its data flows according to a list of threats organized in the mentioned six threat categories. Its result is a list of actual threats for the software system under analysis.

Alice will then perform her analysis following our blockchain threat modeling methodology, composed of four different steps.

Step 1 Assess whether blockchain can be a viable solution at all for the project at hand, in a motto: "when to use the blockchain".

Step 2 Analyze which blockchain implementation can be up to the task, i,e., "choosing a blockchain implementation".

Step 3 Review threats specific to the building blocks of the chosen blockchain solution.

Step 4 Perform a traditional threat modeling assessment for the global solution.

The paper is structured as follows. Section 2 presents the main concepts on blockchains and on threat modeling that are relevant background for our contribution. The four steps of our methodology are covered in Section 3 with an introduction of our methodology, followed by Section 3.1 that presents our criteria to decide about the inclusion of blockchain technologies. Section 3.2 illustrates the different blockchain technologies and their usages, while the associated threats are presented in Section 3.3. The threat modeling analysis of Alice's software project, considering our threat list, is presented in Section 3.4. Lastly, Section 4 shall offer a conclusion.

2. Main concepts and state of the art

This section presents the conceptual underpinnings of our work: a definition of blockchain in its main characteristics, together with a presentation of threat modeling as security architecture review practice.

2.1. Blockchain main concepts

A blockchain can be defined as "a distributed append-only timestamped data structure" [6], that relies on security features like "cryptographic hash, digital signature and distributed consensus mechanism" [10], to achieve its implementation by a peer-to-peer network of nodes without any trust assumption among them. Blockchains essentially record in a *ledger* the *transactions* between two *nodes* of the network: such transactions are signed by at least one of the actors and may involve an exchange of assets (like in crypto-currencies) or not. Once signed, a transaction is transferred to the other nodes, called *verifiers*, that ensure its validity and in this case, the transaction is added to a *block* (can be a set of transactions) and therefore become an official element of the blockchain (appended to the tail of the chain of existing blocks). Crypto-currencies use this mechanism to avoid that the same unit of currency can be spent twice, for example. The decision process of the verifiers is the mentioned "distributed consensus mechanism": multiple options exist and their choice depends on the blockchain typology we are considering. There are essentially two types of blockchain: *private* or *public* access, respectively if nodes are somehow identified or new nodes can be created arbitrarily. Therefore, private blockchain are normally defined *permissioned* as blockchain services are regulated and nodes may use them only if authorized. Conversely, public blockchains are *permissionless* and blockchain network services can be called freely by nodes. In permissionless blockchains, the choice of the consensus mechanism is critical as it governs the transaction validation. Complex mechanisms are needed, robust to attacks like node identity spoofing or forging (like in the Sybil attack [11]). For example, Bitcoin uses the Proof-of-Work method that requires a significant amount of computational resources to discourage this practice, however this choice leads to validation nodes (mining nodes) gathering in pools [12]. On the other hand, in permissioned blockchains, consensus mechanism can be much simpler and faster, with the assumption that nodes are identified and authorized centrally. Last but not least, transactions may be complex, up to comprise a "computerized transaction protocol that executes the terms of a contract" [13]. We talk in this case about *Smart Contracts*

[1] https://www.microsoft.com/en-us/download/details.aspx?id=49168.

and they materialize as source code that is executed when a transaction takes place. Blockchain technologies make extensive use of cryptography for their implementation, also including interactions with end-users. Independently from the type of ledger (permissionless or permissioned), end-users need to be associated with asymmetric keys to operate on the blockchain; therefore, *wallet* software solutions (be them as-a-service or on end-user's local devices) are used to simplify their management.

2.2. Threat modeling

Threat modeling is a practice used to analyze software architectures and discuss their security and functional requirements, to identify potential threats and the associated controls to mitigate them. The STRIDE threat modeling technique [9] was developed at Microsoft since 1999. The acronym STRIDE is composed out of the thread categories considered in the methodology: Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. The methodology, as further analyzed by Scandariato et al. [14], consists of the following steps:

1. Architecture modeling. In this step, the system architecture is analyzed, especially with what concerns the so-called "Data Flow Diagram (DFD)", that highlights the trust boundaries, data storage points, external components etc. Any security assumption, like the availability of an external authentication system, must be declared at this stage.
2. Threat category association. Each element of the DFD gets associated to one or more of the potentially applicable STRIDE threat categories.
3. Threat elicitation. The previously identified associations are analyzed in depth, following a checklist-based approach that points to a number of concrete threats, to come up with a list of applicable specific threats.
4. Documentation. At this stage, the discussion becomes materialized in a document, to allow further analysis and risk assessment, to decide how to mitigate the identified threats (or to accept the risk) and to document the decisions for the following implementation activities.

The effectiveness of the methodology relies on the scope and precision of the analysis, on the experience in security of the team members that is however mitigated by the checklist approach and the list of punctual threats associated to each threat category. For this reason, our contribution comprises a list of blockchain-specific threats that complement those of the basic STRIDE formulation. Other notable techniques to conduct architecture analysis include attack trees [15] and misuse cases [16]. Both methods are valid to identify applicable threats during architecture reviews, however they rely on brainstorming with security experts and on their experience to achieve good results. While not replacing it, STRIDE on the contrary can mitigate a lower security experience in the review team by means of checklists that guide the analysis. For this reason, STRIDE is deemed to be more easily incorporated in industrial secure software development processes and thus to be applied by software developers with security knowledge rather than by security experts.

3. Our methodology

This section presents our methodology, organized in four different steps, illustrated following Alice's design activities.

3.1. Step 1: When to use the blockchain

Technology is useful to solve specific problems, but not all problems can be solved with the same technology. Blockchain is no different. Where it brings useful properties, such as auditability, strong integrity protection and virtual tokens management, there are cases where using blockchain will be detrimental to the problem at hand. This leads to the sensible approach of checking when not to use blockchain at all. In our methodology, we propose to conduct such assessment as the very first step, by running through the workflow depicted in Fig. 1. It caters for a number of key questions for the software architect, leading to an preliminary risk evaluation and initial indications on the blockchain flavors to consider (or not).

Starting from the beginning, the first question to answer is whether the solution requires – or plans to – store personal data into the blockchain itself. A yes should raise a red flag, as once the data is entered, it virtually cannot be deleted. Such situation is clearly not compliant with a number of legal frameworks on data protection around the world (one example for all, GDPR [17]).

> In the case of Alice, she initially thought about storing user profiles (age, gender, country) into the blockchain, namely to compute statistics about players and to do some entertaining advertising. Now, she considers storing profiles in a central database and to keep only the primary key into the blockchain.

Next question in the workflow, is whether the solution requires strong control — in other terms, the possibility to dramatically alter the solution while preventing users of the old solution from using it, or to forcibly redistribute crypto-tokens amongst participants (in this case, the players). This would speak in favor of more traditional solutions — the blockchain was designed for its immutability.

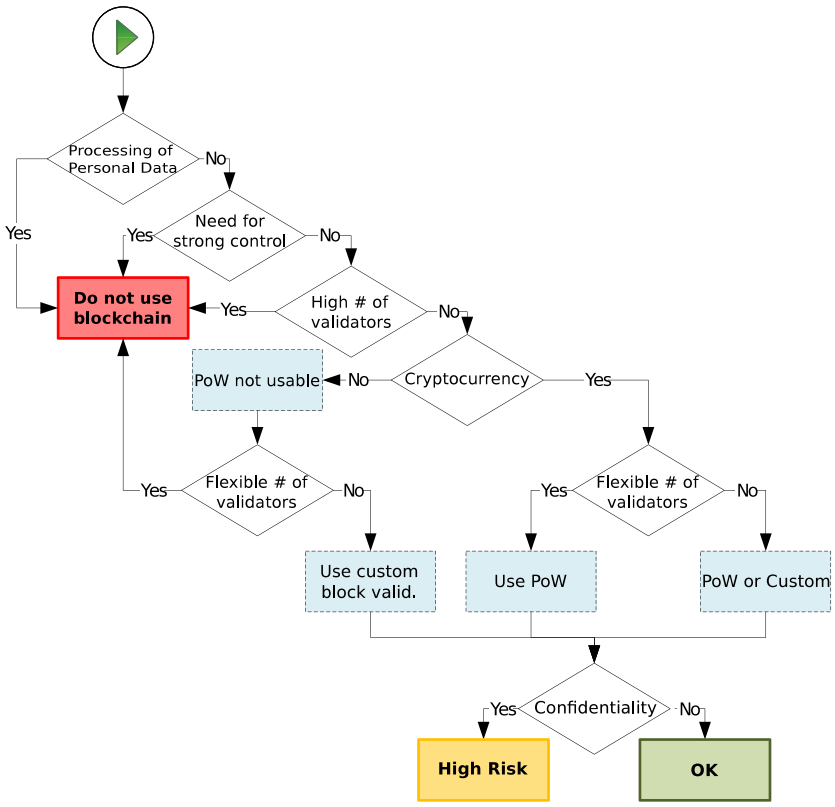

Fig. 1. Decision workflow: when to use the blockchain?

Alice thinks that no strong control will be necessary a priori.

The next question is about the number of validation (e.g. mining) nodes. Too few and the immutability property crumbles.

Alice does not have an opinion on the question at this moment, so she takes note that the number of nodes should be 'sufficient'.

The next check is about existence of a crypto-currency, is closely associated to the way the solution should validate new blocks, i.e., the consensus mechanism. Proof-of-work consensus (originally introduced in [18] and notably part of Bitcoin [4]) is the most robust solution to ensure that nodes cannot collude into creating a fork to take over the blockchain, as it prevents Sybil attacks [19]. However it is very onerous[2] and node owners will run proof-of-work only if they gain more than what they need to invest. Rewarding with a crypto-currency is the most straightforward solution.

Alice wants her players to be able to play regularly. She wants the game to be a freemium, meaning that every day, players who have less than two coins will get their amount set to two coins. Good players will be rewarded by one extra coin every time they beat a high-score. Extra coins should be purchased with other currencies, such as euro or bitcoin. In this regard, the crypto-currency is not supposed to be used as an incentive.

Without a currency as incentive, it is likely that validation nodes will not run proof-of-work. And without proof-of-work, an attacker can perform a Sybil attack to take control. The solution is to use a permissioned blockchain where the number of validation nodes can be controlled. Either all nodes are under control of the solution owner — making the solution a centralized one, or spread amongst trusted partners, requiring extra activity monitoring to detect suspicious forking (in case a partner acts maliciously).

Now Alice needs to decide how to solve the issue of block validation. Since proof-of-work will not be possible in her case, she considers hosting all validating nodes in-house. This should not adversely impact the gameplay, but with a growing number of players, she needs to make sure that the number of validation nodes can scale as well. So at this point, she realizes that maybe her solution will not work at all.

[2] let us consider for example the significant power consumption of the Bitcoin network: https://digiconomist.net/bitcoin-energy-consumption.

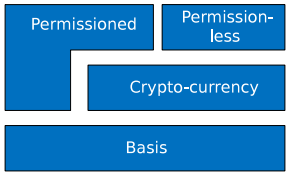

Fig. 2. Different blockchain features.

For the sake of completeness, once the question of the consensus mechanism is solved, the last inquiry is about the requirement to store confidential data into the blockchain. At the difference of personal data, the decision to store confidential data or not can does not have compliance implication and is "only" a security decision. Storing confidential data can be achieved by using encryption. However this opens the door to offline cyphertext attacks, which are likely to remain undetected and which will be virtually impossible to fix.

> Alice thinks about storing user IDs, their scores with the corresponding timestamp and not much more. All this information is public information in her eyes.

At the end of the workflow, the architect has an estimation of the risk and in case, an identification of the blocker(s), like in Alice's case. It can be possible, at this point, to revise some of the answers in order to change the result of the process.

> According to this decision diagram, given the problem of block validation due to the choice of a fitting consensus mechanism, Alice cannot use the blockchain without extra requirements.

For business critical applications, in case of blockers identified for the consensus mechanism, the only option would be to host all validation nodes and to scale as the number of players grows. However, Alice's retrogaming idea is an exploratory project which will not have major adverse consequences if it fails. For this reason, in this specific case, one may want to consider cost-efficient alternatives to proof-of-work consensus mechanism.

> Ultimately, Alice decides to add radical requirements for her solution: the game will be restricted to nodes which can run Proof of Elapsed Time [20], a (relatively recent) consensus mechanism that is not computational intense — thus not requiring a reward for validation nodes, but which requires the usage of an Intel SGX chip on the machine as well as restricting the solution implementation to a blockchain which supports this consensus mechanism, such as Hyperledger Sawtooth [21]. This decision also implies that Alice has to give full trust in Intel and in the security of its hardware and software implementation of the SGX chip. However, this change has impacts on the security design of her solution, as described in the following.

Once the decision to go for a blockchain solution is made, the next step is to consider the impact in terms of security threats bound to the specific, elicited solution implementation requirements. The next section details these implementation details and threats.

3.2. Step 2: Choosing a blockchain implementation

The answers to the decision diagram in Fig. 1 entails requirements for the underlying implementation of the solution. There are two main aspects to be considered, that we define blockchain features and blockchain (-stored) data. Each of them has naturally specific risks associated to their choice, that must be analyzed in the next step of the methodology. When it comes to threat assessment, the idea is to keep in scope all requirements belonging to the selected options.

In terms of features, we see three main options available as illustrated in Fig. 2: permissionless blockchain, permissioned blockchain and crypto-currency (which can coexist with one of the two others). Each of these has dependencies, such as permissionless' reliance on proof-of-work and thus on a crypto-currency, as well as specific security concerns. Architects/Developers desiring to adopt any blockchain solution will also need to review the "basic" risks — the ones generic to all known blockchain implementations.

In terms of stored data (as in Fig. 3), the three options we consider are: hash, generic and smart contract. Hash is, from the security perspective, the safest data type to store in a blockchain. It consists of storing only hashes, which is sufficient if the blockchain is used as an audit component, for example in track-and-trace scenarios or to ensure that certain properties, external to the blockchain, were not modified since the time they were checked in: configuration files or important web application signatures, static content of database tables etc. Generic is for any data which is not a hash, such as arbitrary text. And smart contracts is for data which technically is executable code.

The implementation decisions depend of course on the use-cases of the software project, and are guided by results of the previous *Step 1*. It is important, therefore, to identify the use-cases that are relevant for blockchain implementation and analyze the different options in context.

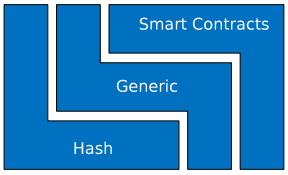

Fig. 3. Options for blockchain-managed data.

Alice's use-case is as follows: first, a player needs to install her/his own game node. If successful, the game node engine will grant to the player two coins per day. These coins will be spent launching a shooter game which will run on the node machine. If the game is successful, Alice plans to add the possibility to have two players play the same game, but at the time the game will be one player only. A transaction shall be sent to the blockchain for each of the following events:

- game started (coin spent)
- 100.000 points earned
- ship destroyed (timestamped)
- game over (timestamped, final score)

In terms of implementation, two options can be considered: either the game is a stand-alone program which communicates with the blockchain, or it runs as part of the blockchain, as a smart contract. At this point, Alice has not decided and will consider the solution which has the overall lowest security risk. In terms of data storage, she also sees two approaches for producing statistics: either the data is correct, or store the data into the blockchain directly. In the 'stand-alone' version, her considered building blocks are as follows: basic, crypto-currency, permissionless, hash. In the 'smart-contract' version, her considered building blocks are as follows: basic, crypto-currency, permissionless, generic, smart-contract.

Now, Alice has two possible implementations in mind. The next section lists all the blockchain threats we have elicited so far, together with Alice's considerations on the threats associated to her project.

3.3. Step 3: Blockchain specific risks

Blockchain-specific risks consist in a list of security, compliance and legal concerns bound to the building blocks considered for the implementation of the solution. As with any technology, the list of risks evolves as new threats are discovered and as new mitigations are devised. For example, today's risk of seeing illegal data being stored in the blockchain (a problem Bitcoin seemed to have fallen victim of[3]), might be mitigated with censorship solutions [23] that are devised and implemented into the technology. This section briefly highlights the 67 threats we currently cover, broken down per building block. Is shall be noted that some of the risks are competing if not directly opposing each other: traceability can be seen as a threat to anonymity, and anonymity can be seen as a threat to traceability. Ultimately, the architect shall decide where to draw the line and which property shall be put forward in his solution.

At the time of writing, we present the list of considered risks. These can further be broken-down into about fifteen categories, omitted in this document, ranging from key management, to privacy, to consensus, to interoperability, to legal concerns etc. Besides the categories mentioned in the previous section, an eighth category, named 'others', was created for risks which are situational. These risks are highly specific to the use-case and should be reviewed in any case, but are more often than not likely to be irrelevant. As a remark, our risks have been elaborated upon discussions triggered by the publication of a report on blockchain by the European agency ENISA [24].

Basis risks (26 risks)

A list of the basis risks can be found in Table 1. Some are connected to key management, like *BAS-1* to *BAS-4* and others to cryptography, *BAS-5* and *BAS-6*. They deal with stealing of wallet or private key, or to the ability of an attacker to forge a private key or even a transaction, resulting respectively in an impersonation of a legitimate user or in the addition of a new transaction into the ledger; normally these situations are connected to weaknesses/flaws in the cryptographic protocols or to the usage of weak keys.

Risks *BAS-7* and *BAS-8* are associated to data protection and privacy. *BAS-7* deals with the need to avoid having personal data stored as public elements (even if in the headers) of the blockchain; under certain regulations, for example GDPR [17], special processing must be ensured for such pieces of information, thus the immutability and publicity of blockchain may stand in the way of compliance. *BAS-8* addresses the possibility that a compromised private key may be used to compromise future transactions associated to that key; forward secrecy is a feature that explicitly prevents such situation.

[3] see [22] and https://news.bitcoin.com/no-isnt-child-porn-bitcoin-blockchain/.

Table 1
Basic blockchain risks.

Risk ID	Risk name
BAS-1	Wallet credential theft
BAS-2	Private key theft
BAS-3	Private key forging
BAS-4	Signature of rogue transaction
BAS-5	Weak key generation software
BAS-6	Resilience of asymmetric keys to 0-days/quantum computing
BAS-7	Data protection & privacy violation (header data)
BAS-8	Lack of forward secrecy
BAS-9	Security vulnerability in the blockchain code
BAS-10	Re-usage of a 'spent' asset
BAS-11	Lack of new blockchain adoption after a hard fork
BAS-12	Lack of means to slow down/stop consensus hijack attempts
BAS-13	Exploitation of wallet to access stored keys
BAS-14	Exploitation of wallet to access keys in transit
BAS-15	Loss of wallet
BAS-16	Lack of detection of wallet duplication
BAS-17	Ledger size too big
BAS-18	Transaction speed too slow
BAS-19	Sharding
BAS-20	Validation bottleneck
BAS-21	Unclear enforcement of legal constraints
BAS-22	Protocol evolution too slow
BAS-23	Untrusted end-user computer (hacked account)
BAS-24	Lack of means to verify the intent of executing a transaction
BAS-25	Exploitation of the transaction protocol (hacked key)
BAS-26	Exploitation of the transaction protocol (non-compliant transaction)

BAS-9 deals with the risk of having exploitable vulnerabilities in the blockchain code, while risks *BAS-10* to *BAS-12* focus on the consensus management mechanism: *BAS-10* foresees the possibility that an asset (like a bitcoin) could be consumed more than once before a transactions is confirmed (i.e., verified) and thus immutable. *BAS-11* is relevant when after an attack, the only way to recover operations is through two measures, an "hard fork" in the chain of transaction to the last not-compromised transaction, and a software update to fix the exploited vulnerability. In this situation, recovery can only be achieved if all nodes accept and update their software timely to reduce service disruptions. *BAS-12* refers to the possibility that consensus attacks may come to alter the standard blockchain behavior and no effective means to counter them are available.

BAS-13 to *BAS-16* refers to wallet management risks, be the wallet hosted locally or remotely. *BAS-13* and *BAS-14* deals with the possibility to obtain blockchain keys exploiting wallet's vulnerabilities. *BAS-15* considers the possibility that an end-user loses her/his wallet, for example because it was hosted on a local device that got stolen or corrupted. *BAS-16* on the other hand anticipates the possibility that a wallet could be stolen and no mechanisms for the detection of such situation are available.

Scalability matters are taken into account in risks *BAS-17* to *BAS-20*. Normally the blockchain ledger is replicated integrally in all nodes, and its size grows indefinitely. *BAS-17* considers that storage capacity may not be adequately planned in advance to cope with the growing ledger size, thus causing disruptions like a denial of service. Transaction speed is the object of *BAS-18* and it must be aligned with the performance requirements of the served use-cases. Sharding (*BAS-19*) occurs when a transaction block needs to be reverted due to an error during block validation, while *BAS-20* considers the case when validation nodes are not able to catch up with the validation requests.

Naturally, blockchain technologies (especially if used in the financial domain) need to incorporate regulatory, anti-fraud and anti-money laundering techniques and mechanisms, therefore *BAS-21* to *BAS-26* look at such elements. First and foremost, *BAS-21* considers the compliance against the applicable legal framework; to counter critical attacks, protocol adaptations or even emergency responses can be needed, so *BAS-22* considers whether such adaptations may be implemented, and in a short time frame. *BAS-23* instead considers cases where an end-user lost control of a device that is subsequently used to conduct attacks or frauds. *BAS-24* deals with non-repudiation, or better, to the possible absence of means to assert that a user intentionally performed a transaction and it did not happen accidentally. On a similar line, *BAS-25* looks at the possibility to exploit a session key or any other key used in the protocol, while *BAS-26* considers vulnerabilities in the protocol flows.

Crypto-currency risks (4 risks)

The list of crypto-currency risks (see Table 2) starts with *CC-1*, connected to the high currency volatility, which in turn may influence negatively on the consensus mechanism when getting too low or on the contrary, cause a surge of transactions if getting very high thus requiring effective scalability. *CC-2* considers the possibility that the crypto-currency gets used to pay malware or any illegal goods, also completely offline. *CC-3* deals with the possible lack of compliance

Table 2
Crypto-currency risks.

Risk ID	Risk name
CC-1	Currency high volatility
CC-2	Usage of crypto-currency to transact malware (side channel)
CC-3	Lack of compliance with anti money-laundering rules
CC-4	Exploitation of the transaction protocol for double-spending

Table 3
Permissionless ledgers risks.

Risk ID	Risk name
PLESS-1	User re-identification via transaction analysis
PLESS-2	Block mining too simple
PLESS-3	Transaction spamming by rogue nodes
PLESS-4	No currency for mining
PLESS-5	Front-running attack
PLESS-6	Lack of means to identify an address owner
PLESS-7	Lack of means to block ongoing illegal transactions
PLESS-8	Power consumption too big

against money-laundry rules coming from the financial domain, while *CC-4* considers vulnerabilities in the transaction protocol to allow double-spending of the same virtual currency.

Permissionless risks (7 risks)

The permissionless ledgers specific risk list (in Table 3) starts with *PLESS-1*, that looks at the possibility to re-identify users' identities by analyzing their transactions; as blockchain does not provide complete anonymity with this respect, this risk should always be taken into account. Naturally consensus management is critical in permissionless blockchain and on top of the risks in the basis section, *PLESS-2* considers the case when block mining is too simple and easy to perform, for example due to the advent of dedicated hardware; in such situation, certain notes may gain unfair advantage and be able to take over the blockchain control and to rewrite a part of the history (the ledger records). Risks associated to denial-of-service attacks are considered in *PLESS-3* and *PLESS-4*. The former looks at attacks performed by rogue nodes triggering transactions (or transaction attempts), if not prevented by protocol or platform measures, while the latter considers the risk that verifiers do not get compensation for their mining activity thus they do not have incentives to perform block validation. This situation may be by default or caused by circumstances, in any case it may be a serious risk for the sustainability of the solution under analysis. *PLESS-5* considers the usage of Smart Contract in permissionless ledgers: normally, transactions are broadcasted to all nodes, giving the basis for front-running attacks where, just before an important buy or a sell takes place, a rogue node submits respectively a buy or sell transaction, in order to exploit the fluctuations caused by the important operation. Such situation may happen for example if transaction processing order is regulated by a combination of rules (like 'biggest amount first', 'biggest transaction fee first') that results vulnerable to this attack. Financial regulations noncompliance is considered in *PLESS-6* and *PLESS-7*, respectively for the lack of means to identify an user (in direct contrast with PLESS-1) or to block a transaction for illegal purposes. Last but not least, *PLESS-8* deals with a scalability risk due to power consumption. Proof-of-Work for validating new blocks costs an increasing amount of energy as there are more nodes validating blocks and as the reward diminishes as it is the case with Bitcoin. There is a risk that the power consumption becomes too big or too expensive, naturally dragging all verifiers to geographic areas where power is cheaper. This in turn concentrates validation nodes closer to each other, opening-up an increased risk of denial of service (in case of localized power outage) or of chain hijack (if a high number of nodes is operated from a few number of network infrastructures and that they are falling under control of an attacker, which may be a cybercriminal or an organization, scaling up to the hosting government).

Permissioned risks (7 risks)

The permissioned legers are operated by a set of known participants and their functionalities are regulated by access control rules. In such context, *PERM-1* deals with the risk that transaction validations is not enough attractive for verifiers, if for example the resources to be committed in Proof-of-Work consensus mechanism are too expensive in comparison with the validation benefits. Again on consensus mechanisms, in *multichain* ledgers, i.e. where it is allowed to have one or more "branches" of the main blockchain (often referred as *side-chains*), *PERM-2* states the risk of not having enough verifiers for guaranteeing the immutability of the side-chains on top of that of the main blockchain. Still on consensus, *PERM-3* and *PERM-4* concentrates on the validation side. As recalled, in general permissioned blockchains are used when the number of trusted verifiers is assumed to be low, which simplifies block validation and consensus mechanism; however it can be possible that an attacker takes over one or more verifiers and at the same time, launches a denial of service attacks on all others in order to rewrite a part of the blockchain. *PERM-3* captures this situation. For the same objectives, attackers may review blockchain policies to find vulnerabilities allowing them to take over on the blockchain

Table 4
Permissioned ledgers risks.

Risk ID	Risk name
PERM-1	Refusal to process a transaction
PERM-2	Lack of incentive to secure a side-chain
PERM-3	Consensus hijack by hack of regulators
PERM-4	Abuse of policy rules for chain takeover
PERM-5	Targeted denial of service for guiding block validation
PERM-6	Disclosure of internal processes

Table 5
Hash data risks.

Risk ID	Risk name
H-1	Hash collision

Table 6
Generic data risks.

Risk ID	Risk name
GEN-1	Decryption of encrypted data
GEN-2	Lack of detection mechanism (repudiation)
GEN-3	Resilience of encryption scheme (confidential data)
GEN-4	Enforcement of the right to be forgotten
GEN-5	Data protection & privacy violation (payload data)
GEN-6	Storage of malicious data

(*PERM-4*). Denial of service for verifiers is the object of *PERM-5*, as also seen in *PERM-3*; in this case, it is contemplated the risk that block validation is delayed, to allow the creation of a fork with such a significant history to become the main chain once the attack terminates (exploiting the blockchain reconciliation procedures). *PERM-6* is about the disclosure of internal processes due to the blockchain transparency properties. Blockchain requests and transactions are signed by the initiating peers. In institutions, where several individual persons all run their own peer, the details of who is involved with each transaction will become visible on the ledger. This might be a confidentiality issue for the institution which may want to act as a unique body (see Table 4).

Hash data risks (1 risk)

In a use-case where only document hashes are being stored as an audit trail of a document repository, *H-1* considers the risk of an attacker forging a document happening to have the same hash value as a genuine one, opening-up the possibility of fraud (see Table 5).

Generic data risks (6 risks)

This category reflects risks connected to the capability of nodes to store arbitrary data onto the blockchain, listed in Table 6. *GEN-1* considers the risk that, as every blockchain node has access to blockchain data payloads, it has access to any encrypted data, so that one can be allowed to perform offline attacks without any kind of limitation or control. As for the attack on encrypted data, *GEN-2* considers that attacks on ledger (offline) copies will be impossible to detect, by definition. Another source of risk is considered in *GEN-3*, with respect to the resilience of the adopted encryption scheme: the advent of quantum computing, for example, may break our current cryptography in some years. Therefore, blockchain data confidentiality should not be considered as guaranteed forever.

GEN-4 and *GEN-5* looks at personal data protection regulations. The former considers the "right to be forgotten" as enunciated by GDPR, i.e., the need for an entity (data controller) to delete all personal data of an individual if she/he requests to do so. Naturally if personal data are stored in a blockchain (or used as identifiers), their deletion conflicts with its immutability property. The latter, the GDPR definition of personal data comprises also IP addresses and other elements that may be part of headers used in transaction or other logged messages. As their processing is strictly regulated (for example, an organization must also enforce the right to be forgotten also on those pieces of information), potential violations of data protection regulations must be assessed also for what concerns headers and any other element of the blockchain. Last but not least, *GEN-6* states that, if arbitrary data can be stored in the blockchain, the mechanism could be abused for example to distribute viruses or illegal content, as it happened with Bitcoin [22].

Smart Contract risks (8 risks)

Smart Contracts can be seen as executable code running as part of blockchain operations. *SC-1* is about a possible re-identification of a person, or information disclosure on internal data processing, because of some data leaking from a Smart Contract implementation either because of a vulnerability or because of a design flaw. On a similar line, *SC-2* consider the possibility that a Smart Contract implementation contains a vulnerability. Since deployed Smart Contracts

Table 7
Smart Contract risks.

Risk ID	Risk name
SC-1	Privacy breach through vulnerability in smart contract
SC-2	Security vulnerability in the smart contract
SC-3	Smart contract-powered denial of service
SC-4	Lack of means to stop a smart contract from running
SC-5	Programming code allows side effects
SC-6	Design flaw in smart contract
SC-7	Zero-day in smart contract code
SC-8	Weak endorsement rules for deployment

cannot be revoked, it is a matter of reducing the attack surface to its minimum. And, since the execution of the Smart Contract is mandatory, in the hypothetical case where the blockchain node can call external URLs, it could possible define a specially crafted contract that connects to an external URL, letting it replicate through nodes and then repeatedly execute it to obtain a distributed denial of service attack (*SC-3*). A problem is also represented by a Smart Contract property, that normally prevents them to be stopped after being deployed (*SC-4*). *SC-5* moreover considers situations where Smart Contracts can be expressed using imperative languages such as C: their execution may have side effects, e.g. a function can modify a variable which is not its return value, thus potentially altering some of the blockchain functionalities. Even worse consequences may happen in case of design flows (*SC-6*) or zero-day vulnerabilities (*SC-7*). Therefore, considering the importance and the sensitivity of Smart Contracts, technologies like Hyperledger Fabric [25] foresee Smart Contract deployments via an endorsement process, to reduce the risk of error. The Smart Contract developer defines two documents along with the Smart Contract: a policy, defining rules by which the Smart Contract abides, and an endorsement policy, containing what specific nodes called endorsers shall abide to. Then, a defined number of endorsers need to authorize the Smart Contract deployment. This shall happen after the endorsers tested the Smart Contract according to the defined policies. However, in this setting, the endorsement rules may be too weak to allow the detection of security flaws (intentional or accidental) (*SC-8*) (see Table 7).

Other risks (situational) (8 risks)

This category (shown in Table 8) is for risks not pertaining to any other category but that are still relevant for blockchain applications. Often blockchain applications rely on cloud platform services,[4] in particular for node hosting and management. *OTH-1* contemplates the possibility that a security vulnerability in the cloud platform may severely affect the network of blockchain nodes. *OTH-2* focuses on a technique to exploit side-chains (when available, as described for *PERM-2*) to consume the same asset more than once; this technique relies on the possibility (if available) to revert a side-chain. This allows an attacker to consume an asset in a side-chain for a transaction, obtain the good or service acquired, and then having the side-chain reverted, so that the asset can be spent again in the main blockchain. Again on side-chains, when one needs to be reverted, complex operations are needed in order to ensure the integrity of the chain(s), that may result in extra computational load on the nodes and thus paving the way towards a denial of service attack (*OTH-3*). Again on potential attacks, *OTH-4* looks at pruning, a technique available in certain technologies which consists in storing only parts of the ledger. However, this feature, if not carefully conceived, might open-up the possibility of making fraudulent blocks look genuine, abusing hash collision. On a completely different perspective, *OTH-5* anticipates a requirement that at this stage seems likely to be developed in the future by the main blockchain technologies, the interoperability between different ledger. Ad-hoc solutions to serve specific use cases are already possible today, for example with the adoption of side-chains to synchronize transactions on both ledgers. Still on interoperability, but from another angle, *OTH-6* considers the actual fragmentation in current wallet offers, as each blockchain requires a different wallet format. *OTH-7* considers situations where a transaction must be reverted, in direct contrast with the immutability property of blockchain. Specific rules might be used if this risk must be considered. Lastly, and again on wallet technologies, *OTH-8* highlights the difficulties associated with wallet addresses. Such addresses are used for transactions and look like long character strings. It will be very difficult for a user to make the distinction between two addresses, opening the possibility to tamper with addresses.

3.3.1. Risk assessment example

Having considered all the risks, Alice is now ready to perform the risk assessment of her two possible solutions, according to the relevant threats. She considers both her 'stand-alone' and 'smart-contract' versions in parallel, and notes the following major points.

With respect to *basis risks*:

- *BAS-1*: users losing their wallet will simply have to forfeit their scores. This should not be such a big deal since a good player is a good player and shall achieve new high scores on a new account.

[4] for example https://www.sap.com/products/leonardo/blockchain.html, https://aws.amazon.com/blockchain/, https://azure.microsoft.com/en-us/solutions/blockchain/.

Table 8
Other (situational) risks.

Risk ID	Risk name
OTH-1	Security vulnerability in the platform code (node-hosting cloud platform)
OTH-2	Fraudulent spending via locking, side-chain transacting, then unlocking
OTH-3	Denial of service upon (big) side-chain revert
OTH-4	Exploitation of pruning mechanism
OTH-5	Lack of possibility to share between different ledgers
OTH-6	Lack of common wallet usage for different ledgers
OTH-7	Transaction revert very hard to achieve
OTH-8	Wallet address hard to visualize

- *BAS-2,BAS-3,BAS-23*: hacked accounts or stolen keys should not be such a big concern either. Users would only lose their current coins. It can be a problem for users having bought a high number of coins with real money. To mitigate that, Alice decides to limit the number of coins one user can possess to a maximum of ten at any time. She also considers the possibility of a reporting feature: a user claiming that his account was stolen shall be given a new account with the amount of previously owned coins, and the hacked account should be banned from appearing in the leaderboard.
- *BAS-7*: with respect to data theft, the user shall only be allowed to enter a screen name, which shall be stored on a 32 character string allowing only letters and numbers. No possibility to enter free-text.
- *BAS-17*: with respect to the ledger size becoming too large, this is a risk Alice is willing to take, hoping for the best.

With respect to *crypto-currency risks*: The tokens shall be used only for running games. A player can have up to 10 coins maximum. Players cannot give coins to one-another and need to buy them with other currencies or wait 24 h to get up to two free coins. Under these conditions, it appears highly unlikely that the coins can be used for malicious activity or have any volatility at all (*CC-1* and *CC-2*).

With respect to the *permissionless risks*:

- Alice decided that her solution shall run proof-of-elapsed-time, the validation of nodes is then distributed to all players who just cannot decide not to validate transactions. Nodes which appear to never participate shall be banned from the game somehow (as measure to counter *PLESS-4*). In the 'smart-contract' version, there can be a parameter checking that the node participated in block validation not too long ago in order to start the game. In the 'stand-alone' version, this might be done by letting the game contact a main server, in turn checking the blockchain status before allowing the game to start. Alice thinks that this solution might be tricky to implement securely…
- *PLESS-1*: user re-identification shall be an accepted risk. When entering a screen name, players will be warned about the risk of using personal information.
- *PLESS-5, -6, -7*: front-running attacks and illegal transactions are out-of-scope as there is no buying nor selling of coins between users.

With respect to the *hash data risk* (for the stand-alone version), Alice is willing to accept it. She will make sure that hash fields are long enough by testing the first prototypes and adjusting the field size as necessary (*H-1*).

With respect to the *generic data risk* (for both versions), only a screen name shall be under the influence of the user (*GEN-6*). All the other fields shall be bound to the application itself. Alice notes that in the stand-alone version, this will require some mechanism (to be considered in the holistic software analysis) to ensure that only data from the game is inserted into the blockchain and that no man-in-the-middle can alter this data. This might be acceptable as the data shall only comprise scores and timestamps, but will require some means to detect players playing 'too good'.

With respect to *smart contract data risks* (for the smart-contract version), what Alice sees as a major concern would be a design flaw (*SC-6*) which would allow an attacker to run a denial of service attack on the game infrastructure, or to overwrite high scores, or to deplete coins of players, or to give himself extra coins. All these cases shall then be monitored by audit rules. In case such a violation is detected, players should simply be warned that the game is rigged and be encouraged to move to a new version. Players could be convinced to migrate by offering free coins on the new game engine.

With respect to the *other risks*, a relevant one could be about side-chain concerns. For the first version, the game shall not rely on side-chains at all, but if the number of player grows significantly, the game might become too slow (*OTH-3*) as events will take time before being stored on the blockchain and side-chains could be a way to address the issue. Side-chains could also be useful for two-player support. Alice simply decides to start simple, it will always be time to reconsider when the time comes.

At this stage, Alice draws the following conclusions:

- both versions will need some monitoring for detecting cheaters
- both versions will need a mechanism for banning users. This might be easier in the smart contract version.

- a flaw in the smart-contract will be harder to fix than on a stand-alone version, as this will require migrating to a new node for all players.
- the game might become too slow if many players play. Side-chains might represent a counter-measure; since side-chains are (often) a specific type of smart contract, this gives a point for a smart contract solution

Ultimately, she now thinks that a smart contract solution would be preferable, as the links between the game and the blockchain will be stronger and harder to tamper. So now what remains to be done is to make sure that the overall solution is sound from an overall security perspective.

3.4. Step 4: Threat modeling assessment

If blockchain is a viable solution for the application at hand, and that after taking into consideration the relevant threats the solution can still fulfill its requirements, one shall review the security of the overall solution from a holistic perspective. The importance of this activity is represented by the fact that many so-called blockchain hacks that were recently discussed in the press, were nothing more than integration failures, such as nodes exposing remote access ports to the Internet [26], websites used for Initial Coin Offering being defaced with wallet addresses belonging to attackers [27], etc. Few were actual blockchain exploits but rather "traditional" vulnerabilities, to be countered with "traditional" measures.

For this last part, an existing threat assessment methodology can be used on the overall architecture, such as the mentioned Microsoft's STRIDE approach [9] whose results will be presented here. The objective of this step in our methodology is to obtain a secure system, considering the blockchain measures integrated with the rest of the software project. Moreover, the results achieved by the execution of our methodology provide contents to the execution of the threat modeling.

In fact, at this stage, an architect has identified (*a*) if blockchain adoption makes sense in the project, (*b*) what blockchain technologies are more suitable for the considered use-cases and (*c*) an in-depth blockchain threat analysis. As recalled in Section 3.2, the STRIDE methodology comprises 4 main steps: architecture modeling, threat category association, threat elicitation and documentation. Our architect can apply the methodology as follows:

- architecture modeling: the results of our *Step 2 and 3* allow to identify the blockchain technology that best suites the project's use-cases, together with other requirements for the overall project: the architect now needs to integrate these technologies with the required functional components in a coherent architecture.
- category association and elicitation: similarly to our *Step 2 and 3*, these phases now consider the blockchain together with the other software components and their integration; however blockchain threat considerations provide a basis for the discussion and thus a significant simplification of their execution.
- documentation: the results of our methodology provide contents on the different aspects of blockchain adoption and materialization in the software project.

Back to Alice's project, we recall that her solution will have the following characteristics:

- it will consist of a game running as a smart contract
- it will rely on a permissioned blockchain
- it will not use or provide any crypto-currency
- it will use as consensus mechanism the Proof of Elapsed Time, thus requiring the integration of the Hyperledger Sawtooth to be run on nodes having a Intel SGX chipset onboard.

Alice now can come up with a software architecture that comprises blockchain and functional components, as depicted in Fig. 4. The architecture is represented using a block diagram as defined in FMC [28]: agents are represented with boxes, storage components with ovals, lines with circles connecting agents represent request–response interactions (with the indicators of the initiators) and simple lines associations between agents.

Alice's architecture is composed by a cloud-served backend that coordinates a network of clients, operated by players. The backend includes the chosen blockchain technology together with the functional components: the former is Hyperledger Sawtooth, represented in the diagram as "Node": it is coupled with the SGX chipset and the ledger, represented as a storage component. The latter is captured by the "Game Manager" actor, namely in charge of the leaderboard, and a "Monitoring Engine" aimed at detecting cheaters. The backend description is completed by a database, containing the "Player Details". On the client side, it is of course a replicated the Hyperledger Sawtooth architecture for the implementation of the smart contract logic, interacting with the player's browser where the game runs.

The STRIDE methodology application lets Alice uncover potential issues:

- A player using the same screen name as another one, leading to the requirement that screen names must be unique
- Tampering of the player details, leading to the decision to register details in a separate process and in defining such content as read-only

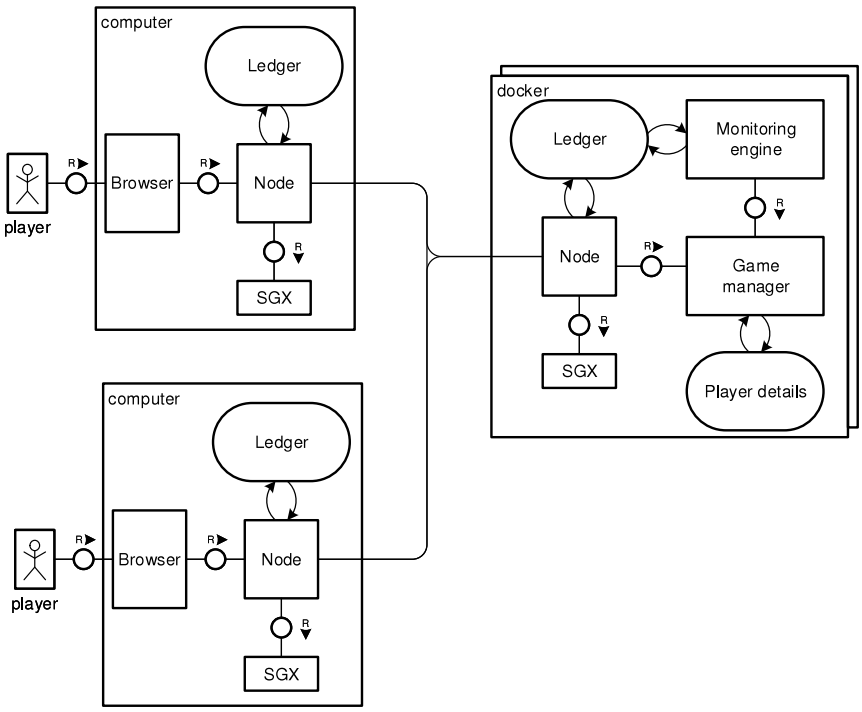

Fig. 4. Architecture: a retrograming distributed application using a distributed ledger.

- A denial of service of the machine running the node connected to the monitoring engine, leading to the decision to host this machine as an auto-scaling docker service

Many more threats may of course be identified. However, and again, they are not blockchain-specific thus out of the scope of this work, but overlooking at them could have disastrous impact on the solution.

4. Conclusions

The general attention to blockchain technologies is due to the valuable security properties potentially achievable through their adoption, for example, immutability, decentralization, transparency and anonymity. As any other technology, they come with constraints that software architects need to consider carefully when designing their software projects. However, the very nature of such properties calls for conscious software design, putting additional strain on software architects that must understand all implications of any design choice. Consequences of wrong decisions may be severe, including attack vectors to compromise transactions or non-compliance with regulations. Therefore, our work aims at helping architects with this important task, providing them with a methodology that allows to assess the security of a software architecture comprising blockchain technologies and guide its secure development. Following our proposed methodology, architects will assess whether the blockchain technologies can be an option at all, given the project's requirements. Subsequently, they can follow what should be a rather familiar threat modeling methodology, STRIDE (that we chose for its popularity and industrial adoption) to analyze their architecture, taking into account our new set of risks that originate from blockchain building blocks. In this way, our methodology allows to identify early security showstoppers in blockchain adoption as well as to highlight blockchain-specific threats. If these threats can be properly mitigated, and if the overall solution is sound after eliciting all its major security risks, the architect will finally obtain a secure-by-design application.

Declaration of competing interest

The authors declared that they had no conflicts of interest with respect to their authorship or the publication of this article.

References

[1] G. Hileman, M. Rauchs, Global Cryptocurrency Benchmarking Study, 33, Cambridge Centre for Alternative Finance, 2017.
[2] J.J. Bambara, P.R. Allen, K. Iyer, R. Madsen, S. Lederer, M. Wuehler, Blockchain: A Practical Guide to Developing Business, Law, and Technology Solutions, McGraw Hill Professional, 2018.
[3] F.J. Hinzen, K. John, F. Saleh, Proof-of-Work's Limited Adoption Problem, NYU Stern School of Business, 2019.
[4] S. Nakamoto, Bitcoin: A P2P Electronic Cash System, 2009.
[5] Bitcoin, Bitcoin Cash Charts, 2018, https://charts.bitcoin.com/bch/. (Accessed 2019-04-04).
[6] F. Casino, T.K. Dasaklis, C. Patsakis, A systematic literature review of blockchain-based applications: current status, classification and open issues, Telemat. Inform. (2018).
[7] R. Valdes, Capture Success From Your Failing Blockchain Project, 2017, https://www.forbes.com/sites/gartnergroup/2017/03/07/capture-success-from-your-failing-blockchain-project/. (Accessed 2019-04-04).
[8] J. Burg, C. Murphy, J.-P. Petraud, Blockchain for International Development: Using a Learning Agenda to Address Knowledge Gaps, 2018, http://merltech.org. (Accessed 2019-04-04).
[9] A. Shostack, Experiences threat modeling at microsoft, in: J. Whittle, J. Jürjens, B. Nuseibeh, G. Dobson (Eds.), Proceedings of the Workshop on Modeling Security, MODSEC08, Held as Part of the 2008 International Conference on Model Driven Engineering Languages and Systems, MODELS, Toulouse, France, September 28, 2008, in: CEUR Workshop Proceedings, vol. 413, CEUR-WS.org, 2008, http://ceur-ws.org/Vol-413/paper12.pdf.
[10] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, H. Wang, Blockchain challenges and opportunities: a survey, Int. J. Web Grid Serv. 14 (4) (2018) 352–375.
[11] T. Swanson, Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems, 2015, https://allquantor.at/blockchainbib/pdf/swanson2015consensus.pdf. (Accessed 2019-04-04).
[12] J. Tuwiner, Bitcoin Mining Pools, 2019, https://www.buybitcoinworldwide.com/mining/pools/. (Accessed 2019-15-04).
[13] N. Szabo, Smart Contracts, 1994, Accessed: 2019-04-04, http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.
[14] R. Scandariato, K. Wuyts, W. Joosen, A descriptive study of microsoft's threat modeling technique, Requir. Eng. 20 (2) (2015) 163–180.
[15] B. Schneier, Attack trees, Dr. Dobb's J. 24 (12) (1999) 21–29.
[16] G. Sindre, A.L. Opdahl, Eliciting security requirements with misuse cases, Requir. Eng. 10 (1) (2005) 34–44.
[17] European Parliament and Council, Regulation (EU) 2016/679 of the European Parliament and of the Council — On the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation), 2016, 27 April 2016 — http://goo.gl/LfwxGe.
[18] C. Dwork, M. Naor, Pricing via processing or combatting junk mail, in: E.F. Brickell (Ed.), Advances in Cryptology — CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16–20, 1992, Proceedings, in: Lecture Notes in Computer Science, vol. 740, Springer, 1992, pp. 139–147, http://dx.doi.org/10.1007/3-540-48071-4_10.
[19] J.R. Douceur, The sybil attack, in: P. Druschel, M.F. Kaashoek, A.I.T. Rowstron (Eds.), Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7–8, 2002, Revised Papers, in: Lecture Notes in Computer Science, vol. 2429, Springer, 2002, pp. 251–260, http://dx.doi.org/10.1007/3-540-45748-8_24.
[20] Intel, PoET 1.0 Specification, 2018, https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html. (Accessed 2018-12-06).
[21] Intel, Hyperledger Sawtooth, 2018, https://sawtooth.hyperledger.org. (Accessed 2018-12-06).
[22] R. Matzutt, J. Hiller, M. Henze, J.H. Ziegeldorf, D. Müllmann, O. Hohlfeld, K. Wehrle, A quantitative analysis of the impact of arbitrary blockchain content on bitcoin, in: Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC). Springer, 2018, In Press.
[23] I. Puddu, A. Dmitrienko, S. Capkun, μChain: How to forget without hard forks, IACR Cryptol. EPrint Archive 2017 (2017) 106.
[24] ENISA, Distributed Ledger Technology & Cybersecurity, Improving Information Security in the Financial Sector, Technical Report 978-92-9204-200-4, 10.2824/80997, ENISA, 2017, Available at: https://www.enisa.europa.eu/publications/blockchain-security.
[25] C. Cachin, Architecture of the hyperledger blockchain fabric, 310, 2016, http://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf. (Workshop on distributed cryptocurrencies and consensus ledgers. Accessed 2019-04-04).
[26] W. Wei, Hackers Stole Over $20 Million in Ethereum from Insecurely Configured Clients, 2018, https://thehackernews.com/2018/06/ethereum-geth-hacking.html. (Accessed 2018-12-06).
[27] M. Kumar, Warning: Enigma Hacked; Over $470,000 in Ethereum Stolen So Far, 2017, https://thehackernews.com/2017/08/enigma-cryptocurrency-hack.html. (Accessed 2018-12-06).
[28] F. Keller, S. Wendt, FMC: an approach towards architecture-centric system development, in: 10th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2003), 7–10 April 2003, Huntsville, AL, USA, IEEE Computer Society, 2003, pp. 173–182, http://dx.doi.org/10.1109/ECBS.2003.1194797.