UDC 004.056

**Y. Baryshev**

# DESIGN PATTERNS SECURITY ANALYSIS FOR BLOCKCHAIN-BASED APPLICATIONS DEVELOPMENT WITH JAVASCRIPT AND SOLIDITY

Vinnytsia National Technical University;

***Анотація***

*Представлено аналіз процесу проектування застосунків із використанням технології блокчейн. Визначено основні загрози застосункам, що базуються на технології блокчейн. Наведено аналіз шаблонів проектування та можливості їх застосування при розробці застосунків на основі технології блокчейн.*

**Ключові слова:** блокчейн, гешування, шаблоні проектування, атаки, протидії.

***Abstract***

*The analysis of the application design process using the blockchain technology is presented. The main threats to the applications based on blockchain technology are identified. An analysis of design patterns and their application possibilities in the development of secure applications based on the blockchain technology is presented.*

**Keywords**: blockchain, hashing, design patterns, attack, counteractions.

## Introduction

Development of blockchain technology provides ability to implement data processing algorithms. In particular, Ethereum is one of the most advanced blockchains from this point of view. Its structure allows appreciating brainchild security of blockchain technology [1], while it is supplemented by both native smart-contract language Solidity [2] and Application Programming Interface (API) for several programming languages, in particular web3.js for JavaScript [3]. This encourages development of sophisticated blockchain-based software. However complexity of an application causes development complexity increasing and the latter one causes programming errors. One should bear in mind, that blockchain technology utilization usually is caused by increased security demands for the designed software product. Recent research results presented in works [4, 5] show imperfection of made design solutions for the most well-known blockchains comparatively to "ideal" cryptographic algorithms. It should ring the bell for the cybersecurity specialists regardless of the fact, that possible attacks are more theoretical, than practical ones, because theoretical attack possibility would be powered by particular design flaws and programming errors.

Design patterns are utilized for complexity management in case of "common" desktop or web applications development. Thus their usage is to be considered for blockchain-based application development focusing on security provided by these patterns.

The research goal is to reduce complexity of blockchain-based application development without security measures losses by the usage of design patterns.

To reach the goal the following tasks are to be solved:

- analyze blockchain-based applications security;
- design patterns comparative analysis according to the criteria;
- develop recommendations towards design patterns implementation.

## Blockchain-based Applications Threats Analysis and Their Counteractions Development

Before analysis of the application development process one should consider design flaws, which had been committed during blockchain development, and thus shouldn't be magnified by application design flaws. The focus of the work was set mostly on Ethereum blockchain as one of the most advanced from the application development point of view, and Bitcoin as the most well-known and widely used one. The abovementioned security analysis results [4] showed similarities between vulnerable to generic multicollision attacks Merkle-Damgård hash construction and blockchain structure despite Ethereum utilizes Keccak hash function [1], which is implementation of sponge hash construction. The point is that Ethereum and Bitcoin implements block connection in the way stated by Merkle-Damgård construction and thus while the very hash function

(Keccack in case of Ethereum) isn't an object of the attack, the block chaining structure is similar to this construction [4]. Moreover the analysis showed, that the transaction hashes within the block are computed using Merkle tree construction, which could be object of the attack proposed in [6]. Several other mistakes were revealed in [5], the most devastating of which is usage of 160 bit addresses, while public key consists of 256 bits. Thus even brute-force preimage attack complexity is $2^{96}$ ($\approx 10^{28,8}$) times reduced [5]. The effect of these flaws could be reduced by implementation of techniques proposed in [7] and storing results within blockchain, but it would result in doubling "blockchaining" (i.e. application data protection) and thus causing increased resources consumption by the application, which is undesirable in the most cases, because of the resources limit and high cost of data storing on the blockchain side (which is implemented by the all blockchain nodes). Therefore the "cryptographic" flaws couldn't be fixed by design patterns in the most cases. However these attacks complexity allows talking about their theoretical implementation, because nowadays from the practical point of view ones implementation couldn't be performed in reasonable time duration [4].

Other threats of application were analyzed such as different DoS attacks according to the classification proposed in [8]. The analysis results shows, that DoS avoiding could be performed by implementation of central/integrating element of an application within blockchain, which is robust to such kinds of attacks. Attacks on the database analysis showed, that combination of the critical data storing at blockchain with multilayer access proposed in [9] would solve most of possible security issues.

## Design Patterns Comparative Analysis and Recommendations

Blockchain-based application usually consists of two major parts: actually blockchain part, i.e. smart-contract written using Solidity [2] for Ethereum blockchain, and user interface written using high-level language such as JavaScript utilizing API [3]. Consequently the analysis was performed bearing in mind these parts. The known software design patterns could be divided into three groups [10]:

- creational;
- structural;
- behavioral.

All design patterns were analyzed according to the abovementioned security analysis results and practical use-cases. The generic design patterns [10, 11] were presented. Results of analysis and yielded recommendations are presented in table 1 [10, 11].

Table 1 – Design patterns and usage recommendation

| Design pattern | Description | Usage recommendation |
|---|---|---|
| Creational patterns | | |
| Factory | Generates an instance for the client without providing any instance logic | Contract deployment, blockchain one-time usage service instance creation |
| Factory method | Delegates the instantiation logic to child classes | Custom parameters contracts deployment |
| Abstract Factory | Factory of other factories | Multiple blockchain interaction instances creation |
| Builder | Creation of particular objects with several parameters avoiding constructors replication | Contract with several parameters creation (multisig for instance) |
| Prototype | Clones object | Only JS part usage/Avoiding for Solidity |
| Singleton | Ensures that only one object of a particular class is ever created | ERC721 and other unique tokens and contracts |
| Structural patterns | | |
| Adapter | Wrap an otherwise incompatible objects | For interfaces creation with already deployed contract |
| Bridge | Compose object, while implementation details are delegated to another object with a separate inheritance hierarchy | User interface creation/Avoiding for Solidity |
| Composite | Lets clients treat the individual objects in a uniform manner | Different similar contract interaction (ERC20 tokens for instance) |
| Decorator | Allows dynamically change the behavior of an object at runtime by wrapping them in an object of a decorator | Avoid |
| Facade | Provides a simplified interface to a complex subsystem | During integration of sophisticated application, different contracts interaction |

Continuation of Table 1

| Design pattern | Description | Usage recommendation |
|---|---|---|
| Flyweight | Shares as much as possible with similar objects | Wise and restrained contracts variables implementation |
| Proxy | Class represents the functionality of another class | For security implementation and complexity management |
| Behavioral Design Patterns | | |
| Chain of Responsibility | Request enters from one end and keeps going from object to object till it finds the suitable handler | Only for JS part |
| Iterator | Presents a way to access the elements of an object without exposing the underlying presentation | Only for JS part/Too expensive for Solidity implementation |
| Mediator | Adds a third party object to control the interaction between two objects | Escrow creation |
| Memento | Storing and retrieving current state of the object | Already implemented within blockchain |
| Observer | Monitoring state changes | Monitoring contract counterparties activity to notify user/other contract |
| Visitor | Separating an algorithm from an object structure on which it operates | Contract interaction |
| Strategy | Allows to switch the algorithm or strategy based upon the situation | Avoid |
| State | Lets change the behavior of a class when the state changes | Implements contracts with "stages" of their existence |
| Template | Define carcass of implementation workflow | Avoid |

The cases, which recommend avoiding usage of certain patterns, don't mean these patterns inefficiency in general, but merely seemed to cause security issues at certain circumstances during blockchain application development.

## Conclusion

Performed analysis of blockchain-based application development shows increased requirements to their security. The instances of such threats were presented for cases of privacy, integrity and availability violation. It was determined, that some privacy issues, those had arisen because imperfection of the blockchain technology implementation, are nearly impossible to fix. However, certain counteractions were proposed for the all abovementioned attack vectors.

Design patterns analysis shows, that the most part of well-known patterns could be used during blockchain-based application development.

## REFERENCES


1. G.Wood Ethereum: A Secure Decentralised Generalised Transaction Ledger, 32 p. URL: http://gavwood.com/paper.pdf (accessed 10.03.2019)
2. Solidity: Language Documentation. URL: https://solidity.readthedocs.io/en/v0.5.5/index.html#language-documentation (accessed 10.03.2019)
3. web3.js - Ethereum JavaScript API. URL: https://web3js.readthedocs.io/en/1.0/ (accessed 10.03.2019)
4. K. Halunen, V. Vallivaara, A. Karinsalo. On the Similarities Between Blockchains and Merkle-Damgård Hash Functions // Proceedings of IEEE 18th International Conference on Software Quality, Reliability, and Security Companion, 2018, p. 129-134.
5. Баришев Ю. В. Аналіз стійкості технології блокчейн на прикладі реалізацій Bitcoin та Ethereum // XLVII Науково-технічна конференція підрозділів Вінницького національного технічного університету, Вінниця, 2018, 3 с. URL: http://ir.lib.vntu.edu.ua/bitstream/handle/123456789/20535/4351.pdf?sequence=3 (дата звернення 10.03.2019).

6. J. J. Hoch, A. Shamir. Breaking the ICE – Finding Multicollisions in Iterative Concatenated and Expanded (ICE) Hash Functions // International Workshop on Fast Software Encryption, 2006, p. 179-194. URL: https://link.springer.com/content/pdf/10.1007%2F11799313_12.pdf (accessed 10.03.2019).

7. V. Luzhetskyi, Y. Baryshev. Methods of generic attacks infeasibility increasing for hash functions // Proceedings IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), Berlin, 2013, p. 661-664.

8. O. Voitovych, Y. Baryshev, E. Kolibabchuk, L. Kupershtein. Investigation of simple Denial-of-Service attacks // Problems of Infocommunications Science and Technology (PIC S&T), Kharkiv, 2016. p. 145-148.

9. O. Voitovych, L. Kupershtein, V. Lukichov, I. Mikityuk. Multilayer Access for Database Protection // International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, 2018, p. 474-478.

10. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Toronto: Addison-Wesley, 1995, 397 p.

11. A. Kamran et al. Design Patterns for Humans! https://github.com/kamranahmedse/design-patterns-for-humans/blob/master/README.md (accessed 10.03.2019)

***Баришев Юрій Володимирович*** — канд. техн. наук, докторант кафедри захисту інформації, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: yuriy.baryshev@gmail.com

***Baryshev Yurii V.*** — PhD. (Eng), Post-doctoral Student of Information Protection Department, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, email : yuriy.baryshev@gmail.com