Towards the Identification of Security Tokens on Ethereum

Monika di Angelo, Gernot Salzer
Faculty of Informatics, *TU Wien*, Favoritenstraße 9-11, 1040 Vienna, Austria
{monika.di.angelo,gernot.salzer}@tuwien.ac.at

***Abstract*—Crypto tokens are digital assets, similar to the coins of a cryptocurrency, except that they do not have their own blockchain or distributed ledger. Rather, they are built on top of an existing one. Areas of application include their use as means of investment, as a local currency in a decentralized application, as well as means for building an ecosystem or a community. Depending on the purpose, it is common to categorize tokens into payment tokens, security tokens and utility tokens. The distinction is of interest since in most jurisdictions, security tokens are more heavily regulated than other tokens.**

In this paper, we present a heuristic approach towards automatic detection of security tokens from blockchain data. To this end, we first discuss several methods for the (semi-) automatic identification of token contracts. Then we attempt to identify the token type. For our analysis, we examine both the deployed bytecode and the calls to token contracts that we extract from transaction data of the Ethereum main chain up to block 9500000, mined on Feb 17, 2020.

***Index Terms*—analysis, Ethereum, smart contract, token, transaction data**

I. Introduction

Tokens are commonly seen as the killer application of blockchains and cryptocurrencies. They can act as a medium of exchange similar to a currency. When integrated into a distributed application, tokens can serve as a local currency. The most outstanding feature of tokens is their programmability as they are managed by smart contracts. These are small programs that run on a peer-to-peer network with the purpose of automating the exchange of digital assets without the need for an external trusted authority. Such assets may be linked to non-digital objects or values.

Crypto tokens have started to change financial processes. Smart contracts facilitate the combination of tokenized assets with coded rules that are automatically enforced. Thereby they cause a shift in trust towards a technology that still has some challenges to solve [1]. Nevertheless, companies already use cryptocurrencies and tokens for funding or enhancing services. Similarly, decentralized finance (DeFi) and FinTech[1] are booming. The increase in the use of digital assets has alerted governments and stakeholders to seek clarity with respect to regulations, especially whether assets qualify as securities.[2]

[1]DeFi refers to the ecosystem comprised of financial applications developed on top of blockchain systems, while FinTech refers to the integration of digital and online technology into financial services.

[2]The relevance of this question can be hardly overstated: On April 3, 2020 eleven class action complaints were filed in New York, USA, against major crypto-companies alleging the sale of unregistered securities.

This paper has been accepted at NTMS-BSC 2020-2021.

A high-level categorization of crypto tokens distinguishes between payment tokens, security tokens, and utility tokens [2]. The distinguishing feature is the investment purpose of security tokens, while utility tokens typically serve the functioning of a product. Payment tokens fulfill a payment role with little or no other function. Classifying a token reliably along these lines can turn into a difficult task, as it may require legal expertise and negotiations with regulatory bodies.

We aim at a better understanding of the use and potential of token contracts by analyzing them on a large scale. Due to their number (over 200 000 on Ethereum), classifying them manually on an individual basis is not feasible. An interesting option is to infer their types automatically, at least with some level of confidence.

In this paper, we present first steps towards this goal. We focus on Ethereum as the major platform for token contracts, with plenty of data available. Our method analyzes the bytecode of token contracts and extracts characteristic metrics for their interfaces. We hypothesize that these numbers predict to some degree the token type. The qualitative evaluation is based on six token contracts under review by regulatory bodies and nine further contracts from the main chain. Our work contributes to the field of crypto asset analytics by providing a method for their detection and classification.

Roadmap. Section II clarifies definitions of token types in several jurisdictions, while section III presents implementation standards and the data. Our methods for identifying token contracts and categorizing the types are elaborated in section IV. We apply the type distinction in section V and show exemplary tokens in section VI, before we conclude in section VIII.

II. Types of Tokens

In general, security tokens are more strictly regulated than other types, with their definition varying among different jurisdictions. Work like [2]–[5] discusses legal and functional differences of token types, where the features of a specific token determine the assessment.

A. Swiss Definition

Security Tokens [6] are "assets, such as a debt or equity claim on the issuer. In terms of their economic function, therefore, these tokens are analogous to equities, bonds or derivatives." Typically, it is a share in the issuing company (equity token).

Utility Tokens [6] are usually backed by a project or application with a definable benefit (like access) and intend to "provide

access digitally to an application or service by means of a blockchain-based infrastructure. The issue of utility tokens does not require supervisory approval if the digital access to an application or service is fully functional at the time the tokens are issued." The purpose of a utility token may include voting rights, some sort of reward, or staking governance.

B. US Definition

The US Supreme Court established the Howey test for determining the existence of an investment contract (security) [5]. It checks four criteria: (1) The investment of money (2) in a common enterprise (3) with the expectation of profits (4) solely from the efforts of others. The US Securities and Exchange Commission (SEC) clarified its application to digital assets in [7], stating that conditions (1) and (2) will typically be met, while (3) and (4) are the distinguishing aspects. SEC also noted that the Howey test is flexible "to meet the countless and variable schemes devised by those who seek the use of the money of others on the promise of profits" [5]. Crypto tokens that are only useful within an application (like game tokens) are less likely to be considered securities.

C. EU Definition

The Market in Financial Instruments Directive (MiFID II, directive 2014/65/EU) [8] is central to EU legislation concerning securities. "Generally speaking, EU securities regulation adopts a territorial and market-focused approach ... territoriality is given a broad reading, focusing on the effects on EU markets rather than the place where actions and omissions take place." [3]. For securities under EU law, there are three formal criteria plus a substantive one: (1) transferability, (2) standardization, (3) negotiability on capital markets, and (4) comparability to a list of examples. Again, the assessment of a token depends on the provided features.

III. Implementation Standards and Data

As a digital asset on top of a cryptocurrency, a token is usually implemented as a smart contract.

Abstract Binary Interface (ABI). Most contracts on Ethereum adhere to the ABI standard [9]. It identifies functions by signatures that consist of the first four bytes of the Keccak-256 hash of the function name together with the parameter types. Thus, the bytecode of a contract contains instructions to compare the first four bytes of the call data to the signatures of its functions. The presence of a particular function in a contract can be checked by locating the corresponding 4-bytes hash in its deployed bytecode. Thus, the compliance of a contract with interface standards can be determined via its bytecode.

ERC-20 Token Standard [10] is the most widely used and most general token standard that "provides basic functionality to transfer tokens, as well as allows tokens to be approved so they can be spent by another on-chain third party." It lists six mandatory and three optional functions as well as two events to be implemented by a conforming API.

Data. We base our analysis on static and dynamic transaction data and execution traces of the Ethereum main chain up to block 9 500 000 (mined on Feb 17, 2020).

The static data consists of the deployment address, the bytecode, its skeleton, the list of the extracted 4-byte signatures of the implemented functions, and a flag indicating whether the bytecode implements an event related to token transfers. Moreover, we use the function headers and contract names where available.

The dynamic data consists of all calls to and from contracts (including the signature of the invoked function) as well as the signatures of the events actually emitted. The dynamic data is sparse: for most contracts, only a small fraction of the offered functions has ever been called, and many events have never been emitted. Moreover, observing a call to a contract with a particular signature does not mean that the corresponding function is indeed implemented; often a so-called fallback function catches unknown signatures without raising an error. Only if a function is frequently called, it is safe to assume that it is part of the interface. To get a more complete picture, we accumulate the dynamic data for all contracts with the same bytecode. If some behavior is observed with a particular contract, then we may assume that every contract with the same bytecode will behave identically.

IV. Identification of Token Contracts

Token contracts offer standardized interfaces so that applications like wallets and exchanges can interact with them. Our pattern-based tool [11] extracts the interface in the form of 4-byte function signatures from the bytecode. Then, we try to recover the function headers from the signatures in order to understand the purpose of the contracts. Finally, based on function name and argument structure, we classify the headers into token-related, neutral, and other functions.

A. Extraction of Function Signatures

When calling a contract that adheres to the ABI standard, the first four bytes of the call data identify the function to be executed. The bytecode of the contract compares these bytes to the signatures of the implemented functions and branches to the respective code. Depending on the tool that generated the bytecode – most commonly some version of the Solidity compiler – the comparison is done in a variety of ways.

For extraction, algorithm 1 uses eight pairs of regular expressions, where the first expression in each pair locates the code that reads the call data, and the second one is applied repeatedly to extract the signatures from the comparisons.

For example, the extraction algorithm yields the signatures $\{\texttt{0x41c0e1b5}, \texttt{0x6b590248}, \texttt{0xecfc0073}\}$ for the contract created at the Ethereum address `0x776f55fa27644705156a46e8c1b2dc28ca122832`.

Evaluation: To validate our algorithm, we applied it to the bytecodes of 81 k verified contracts from `etherscan.io`, using the ABIs given there as ground truth. The signatures extracted by our tool differed from the ground truth in 71 cases. Manual inspection revealed that our tool was correct

Algorithm 1 Extracting signatures from bytecode.

$\text{REDATA}_1 \coloneqq \langle$*regular exp. for first four bytes of call data*$\rangle$
$\text{RESIG}_1 \coloneqq \langle$*regular exp. for signature and returning it*$\rangle$
... 7 more pairs $(\text{REDATA}_i, \text{RESIG}_i)$ ...
function EXTRACTSIGNATURES(*code*)
  code = REMOVEDATASECTION(*code*)
  sigs = $\emptyset$
  for (*reData*, *reSig*) in $[(\text{REDATA}_1, \text{RESIG}_1), \dots]$ **do**
    c = *code*
    if *reData* matches *c* **then**
      c = REMOVEMATCHEDPART(*c*)
      while *reSig* matches *c* **do**
        sigs = *sigs*$\cup\{\langle$*signature returned by reSig*$\rangle\}$
        c = REMOVEMATCHEDPART(*c*)
    if *sigs* $\neq \emptyset$ **then**
      break
  return *sigs*

also in these cases, whereas the ABIs on Etherscan did not faithfully reflect the signatures in the bytecode (e.g. due to compiler optimization or library code).

B. Restoration of Function Headers

To understand the purpose of a contract, we recover the function headers from the signatures. As the latter are partial hashes of the former, we use a dictionary of 340 k headers with their 4-byte signatures (collected from various sources), which allows us to obtain a function header for 59 % of the 278 k distinct signatures on the main chain.[3] Since signatures occur with varying frequencies and codes are deployed in different numbers, this ratio increases to 89 % when picking a deployed contract at random. For the sample contract in section IV-A, our dictionary translates the signatures `0x41c0e1b5` and `0x6b590248` to the headers `kill()` and `getDigit()`, whereas the header for `0xecfc0073` remains unknown.

C. Classification of Function Headers

To gain insights into the use of token contracts at large, in particular regarding the distinction of security vs. utility tokens, we group the functions offered contracts by their purpose. A precise classification in large quantities would require an automated code analysis to check for semantic properties, which is a difficult problem not yet adequately solved. Instead, we present a heuristic test that starts from the observation that the functions of token interfaces can be categorized into the following three groups.

The token-related group comprises the core functions mandated by the standards, as well as related functions to create, destroy and distribute tokens. The second group contains the neutral functions that can appear in any type of contract, like

[3]An infinity of possible function headers is mapped to a finite number of signatures, so there is no guarantee that we have recovered the original header. The probability of collisions is low, however. E.g., of the 340 k signatures in our dictionary only 20 appear with a second function header.

TABLE I
THREE OF 22 CLASSIFICATION RULES

rule	REIN_i	REEX_i	LABEL_i
1	`^(get|is|total|balance)`	`^issue`	`getter`
	If header starts with 'get', 'is', 'total', or 'balance', but not with 'issue', then label it as 'getter'.		
2	`ico`	`unicorn|icoin`	`ico`
	If header contains 'ico' but neither 'unicorn' nor 'icoin', then label it as 'ico'-related.		
3	`icoinfo`		`ico`
	If header contains 'icoinfo', then label it as 'ico'-related.		

getters and setters for public variables or role management. The third group consists of the remaining functions, which may rely on tokens but are not necessary for operating them.

Algorithm 2 classifies functions according to their name, applying the rules shown in table I. In general, we cannot expect to understand the purpose of contracts by just looking at the function names. However, the names in the first and second group are quite uniform and stereotypical as the functions perform standardized tasks. Therefore, this heuristic seems worthwhile in the context of token contracts.

Algorithm 2 Classifying a function header regarding its purpose.

$\text{REIN}_1 \coloneqq \langle$*regular exp. for inclusion of header*$\rangle$
$\text{REEX}_1 \coloneqq \langle$*regular exp. for exclusion of header*$\rangle$
$\text{LABEL}_1 \coloneqq \langle$*label identifying class*$\rangle$
... 21 more triples $(\text{REIN}_i, \text{REEX}_i, \text{LABEL}_i)$ ...
function CLASSIFYHEADER(*header*)
  classification = $\emptyset$
  for (*rI*, *rE*, *l*) in $[(\text{REIN}_1, \text{REEX}_1, \text{LABEL}_1), \dots]$ **do**
    if *rI* matches *header* and *rE* does not match *header* **then**
      classification = *classification* $\cup \{l\}$
  return *classification*

Limitations: The effectiveness of this method hinges on how carefully the rules are chosen. Moreover, the method assumes that for the first and second group, the names of functions indicate the implemented functionality. Finally, 6 % of ERC20-compliant tokens delegate some function calls to another contract (like a library) such that the signatures extracted from the contract represent only part of the interface. To simplify our analysis, we neglect such contracts for the time being.

V. CLASSIFYING TOKEN CONTRACTS

We propose a heuristic method that assesses the purpose of a token contract by analyzing its interface. It uses algorithm 2 to partition function headers into the three groups 'token-related', 'neutral' and 'other'. We call a token contract *pure* if its interface exclusively consists of functions of the first two groups, 'token-related' and 'neutral'. For a pure token contract, our method finds no evidence that it offers a genuine service or product on-chain; it thus may be a security token. Non-pure

tokens, on the other hand, are more likely to be utility tokens, implementing a service using their 'other' functions.

Our method considers contracts in isolation, disregarding companion contracts and off-chain components. A pure token might in fact be part of a decentralized application that, as a whole, offers a service. A reliable classification therefore requires a manual analysis of the context in which the token contract operates.

Figure 1 shows the 188 k deployments of ERC-20 compliant tokens on Ethereum over time, where the upper horizontal axis indicates the date line, while the lower one shows the Ethereum block number. Each bar represents the accumulated number of deployments within 100 k blocks (about two weeks). After a hype in the first half of the year 2018, ERC-20 deployments seem to stabilize at about 1000 per week.

The 101 k distinct ERC-20 bytecodes (deployed 188 k times) contain 55 k distinct signatures, of which 42 k can be decoded to function headers. Because of their uniformity, we expect signatures from the first two groups, 'token-related' and 'neutral', to be mostly among the decodable ones. This assumption can also be justified ex-post by the large number of tokens that turn out to be pure tokens, as any undecoded signature would show up as 'other'.

We specified 22 rules (cf. table I) that divide 35 k function headers into 17 categories, with 7.5 k remaining unclassified. We regard headers of the categories *token*, *distribution*, *auction*, *minting*, *approval*, *kyc*, *ico*, *transfer*, *crowdsale*, *airdrop*, and *burning* as token-related (first group), whereas the categories *control*, *math*, *getter*, *setter*, *trading*, and *roles* count as neutral (second group). 13 k undecodable signatures and 7.5 k unclassified functions form the other (third) group.

Interestingly, 76 k distinct bytecodes (corresponding to 145 k deployments) implement only token-related and neutral functions. According to our definition, they are pure tokens that do not implement a recognizable service or product, and therefore could be security tokens.

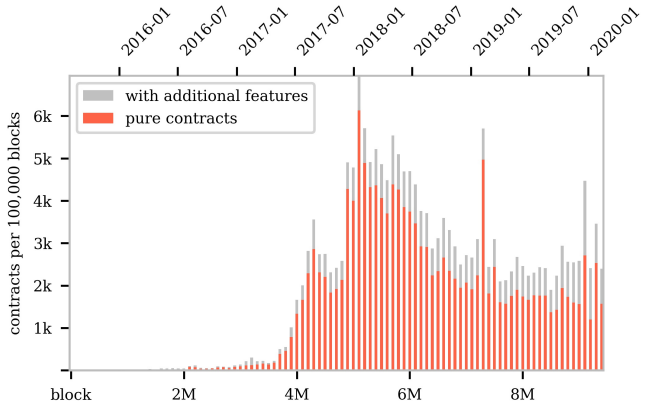

Fig. 1. Deployment of pure (red) and non-pure (grey) ERC-20 token contracts. The lower horizontal axis indicates the Ethereum blocks, while the upper axis shows the corresponding dates. Each bar represents a bin of 100 000 blocks corresponding roughly to 2 weeks.

The remaining 25 k bytecodes implement other functions as well. As this group contains 20 k other signatures, it is not apparent how to decide automatically whether they offer a genuine service or product. A different approach is needed to classify further functionalities of token contracts.

VI. Sample Tokens

The SEC[4] assessed the tokens GLA[5], KIN[6], BQ[7], and CTR[8] as securities, while it issued a no-action letter for the tokens TKJ[9] and Q2[10]. (The Ethereum addresses appear as footnotes.)

GLA: Gladius token is linked to a DDoS protection service. The company reported itself and refunded the proceeds from the ICO in 2017 to avoid a fine from the SEC.

KIN token is linked to a social media platform. In mid 2019, the token migrated to its own mainnet and the SEC filed a complaint against Kik Interactive for their ICO in 2017.

BQ: Bitqy token refers to a market place. The SEC settled with Bitqyck after an alleged fraud.

CTR: Centra token refers to financial services. All three co-founders were indicted for fraud.

TKJ: Turnkey Jet token is linked to air charter services.

Q2: Quarter token is linked to playing video games.

TABLE II
Assessment of Example Tokens (SEC)

name	related	neutral	other	holders	transfers	market cap
GLA	12	5	0	9 552	23 685	0.6 mio
KIN	14	3	0	49 181	499 167	57.0 mio
BQ	9	4	0	20 824	205 212	35,042
CTR	12	22	6	15 966	96 526	1.6 mio
TKJ	16	0	0	1	2	0
Q2	17	11	5	259	470	0

Table II shows the number of related, neutral and other functions for these tokens. We also included, from `etherscan.io`, the number of token holders, token transfers and the fully diluted market cap on March 14, 2020. For a security token, we expect the number of other functions to be 0. In case it is greater than 0, we have to inspect these functions. For the three tokens GLA, KIN and BQ the crucial value is 0. For the token CTR, the six other functions all refer to cards (e.g. `cards_gold(uint256)`), which fits the token's purpose as financial service. Regarding the non-security tokens TKJ and Q2, we expect the number of other functions to be greater than 0. This is only the case with Q2, while TKJ does not provide a service or product on the chain. Even though we have no data about the off-chain services that are linked to TKJ, we can determine that TKJ is not used for whatever use case it should support.

[4] https://www.sec.gov/search/search.htm
[5] `0x71d01db8d6a2fbea7f8d434599c237980c234e4c`
[6] `0x818fc6c2ec5986bc6e2cbf00939d90556ab12ce5`
[7] `0xF0f8B0B8DBB1124261FC8d778E2287e3Fd2Cf4f5`
[8] `0x96A65609a7B84E8842732DEB08f56C3E21aC6f8a`
[9] `0x3c46b50b4d8ba7d5e9a1083f17424a843b8aedfc`
[10] `0xc3a36fad9d3e87acbc69bcd06850dbf13db2ab59`

As a further small ground truth, we analyzed a few known token contracts of which the results are shown in table III. The upper four examples provide diverse services that are not reflected in the respective token contracts (as the number of other functions is 0 to 1). In contrast, the lower five examples are games that implement at least part of the application in the corresponding token contract (6 to 47 other functions). Games are a typical application category hence we included game tokens in the sample.

TABLE III
ASSESSMENT OF EXAMPLE TOKENS (GAME)

related	neutral	other	name	purpose
12	6	0	DxToken	platform for computing
16	6	1	Dragon	payment for entertainment
14	4	0	MANAToken	marketplace (Decentraland)
18	7	0	SNT	wallet app (Status)
9	7	13	Centurions	Crypto Rome
13	18	6	Etheremon	Ether monsters
11	4	7	CryptoSaga	RPG
10	10	47	HyperDragons	strategy battle game
18	7	46	Gods Unchained	eSports

VII. COMPARISON TO RELATED WORK

To detect code clones, the authors of [12] first deduplicate contracts by "removing function unrelated code (e.g., creation code and Swarm code), and tokenizing the code to keep opcodes only". Then they generate fingerprints of the deduplicated contracts by a customized version of fuzzy hashing and compute pair-wise similarity scores. In another approach to clone detection, the authors of [13], [14] characterize each smart contract by a set of critical high-level semantic properties. Then they detect clones by computing the statistical similarity between the respective property sets.

To detect token systems automatically, the authors of [15] compare the effectiveness of a behavior-based method combining symbolic execution and taint analysis, to a signature-based approach limited to ERC20-compliant tokens. They demonstrated that the latter approach detects 99 % of the tokens in their ground truth data set.

Our method of computing code skeletons is comparable to the first step for detecting similarities by [12]. Instead of fuzzy hashing as a second step though, we rely on the set of function signatures extracted from the bytecode and manual analysis, as our purpose is to identify token contracts reliably. While the usage of signatures is in line with [15], we extend it with methods for automatic type inference.

VIII. CONCLUSIONS

Determining the purpose and the legal status of a token contract in a reliable manner is intrinsically difficult and will remain the realm of experts. Clearly, it is infeasible to classify large quantities of contracts this way. A statistical analysis of all contracts, for gaining e.g. insights into the use of contracts at large, has to rely on other approaches.

In this paper, we propose a heuristic method for identifying pure tokens automatically, by checking that their interface offers only token-related and neutral functions. Pure tokens show no sign of a genuine product or service and thus may be security tokens. Of the 188 k contracts on the Ethereum mainchain (as of Feb 2020) that comply with the ERC-20 standard, 77 % are pure tokens.

For a qualitative evaluation, we compare the results of our method to the SEC ruling on six tokens. Moreover, we apply our method to nine further tokens after classifying them manually as security or utility token. Our method yields meaningful results that indicate that the prototype, after extending and refining the classification rules for function headers, is suited for a statistical analysis.

Even though effective in the case of standardized token contracts, deriving the purpose of a function from its header is a rather crude heuristics. Moreover, it presupposes that the header is actually known. A more general approach requires semantic code analysis to identify behavior typical of certain applications, as well as tools able to perform it automatically on large numbers of contracts.

REFERENCES

[1] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[2] L. Oliveira, L. Zavolokina, I. Bauer, and G. Schwabe, "To token or not to token: Tools for understanding blockchain tokens," in *International Conference on Information Systems (ICIS)*. AIS eLibrary, 2018.

[3] P. Hacker and C. Thomale, "Crypto-securities regulation: Icos, token sales and cryptocurrencies under eu financial law," *European Company and Financial Law Review*, vol. 15, no. 4, pp. 645–696, 2018.

[4] J. Rohr and A. Wright, "Blockchain-Based Token Sales, Initial Coin Offerings, and the Democratization of Public Capital Markets," *Hastings LJ*, vol. 70, 2019.

[5] M. Mendelson, "From Initial Coin Offerings to Security Tokens: A US Federal Securities Law Analysis," *Stan. Tech. L. Rev.*, vol. 22, 2019.

[6] FINMA, accessed 2019-10-12. [Online]. Available: https://www.finma.ch/en/documentation/dossier/dossier-fintech/entwicklungen-im-bereich-fintech/

[7] SEC, accessed 2020-02-01. [Online]. Available: https://www.sec.gov/corpfin/framework-investment-contract-analysis-digital-assets

[8] European Parliament and the Council of the European Union, "MIFID II: directive 2014/65/EU," accessed 2020-01-20. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014L0065&from=EN

[9] "Contract ABI Specification," 2019, accessed 2019-09-09. [Online]. Available: https://solidity.readthedocs.io/en/latest/abi-spec.html

[10] F. Vogelsteller and V. Buterin, "ERC-20 token standard," 2015, accessed 2019-10-12. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20

[11] M. Di Angelo and G. Salzer, "Tokens, Types, and Standards: Identification and Utilization in Ethereum," in *Int.l Conf. Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2020.

[12] N. He, L. Wu, H. Wang, Y. Guo, and X. Jiang, "Characterizing code clones in the ethereum smart contract ecosystem," *arXiv:1905.00272*, 2019.

[13] H. Liu, Z. Yang, C. Liu, Y. Jiang, W. Zhao, and J. Sun, "Eclone: Detect semantic clones in ethereum via symbolic transaction sketch," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. ACM, 2018, pp. 900–903.

[14] H. Liu, Z. Yang, Y. Jiang, W. Zhao, and J. Sun, "Enabling clone detection for ethereum via smart contract birthmarks," in *IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE Press, 2019, pp. 105–115.

[15] M. Fröwis, A. Fuchs, and R. Böhme, "Detecting token systems on ethereum," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019.