An Enhanced Blockchain-Based Digital Forensics Architecture Using Fuzzy Hash

Luka Boljević
lb7093@student.uni-lj.si

Teodor Janez Podobnik
tp7220@student.uni-lj.si

Hana Zlobec
hz2731@student.uni-lj.si

ABSTRACT

A recently proposed way to achieve better IoT security using blockchain technology includes an immutable ledger, a decentralization architecture and a well-established low-level cryptographic algorithm. An issue that presents itself is that not all IoT devices are compatible with existing implementations of this idea, as some of them use conventional hash, which is useless for comparing similar files (as even the smallest change causes the hash to be completely different). The authors of the reviewed paper [5] propose an approach using fuzzy hash (in addition to conventional hash for authentication) to construct the Blockchain's Merkle tree, which allows the discovery of altered files by comparing its hash to the one in the Blockchain network. We propose and implement an improved approach, which could be beneficial to all types of digital forensics.

Keywords

Blockchain, Internet of Things, Fuzzy hash, IoT forensics, digital examination

1. INTRODUCTION

Everyday usage of IoT devices is becoming more and more common, which means that the diversity and sheer amount of data that forensic tools have to process in case of an investigation is also rapidly increasing. IoT stands for "**The Internet of Things**" and is a network of physical objects with sensors, processing ability, ability to exchange data with other devices and does not require human interaction to do so. A common examples of IoT devices are "smart home" appliances e.g. thermostats, smart locks, cameras... but also drones and cars connected to the network, which creates a wide attack surface that expands with more such devices emerging.

The diverse nature of data from IoT devices provide difficulties for ensuring the collected evidence has not been tampered with. This is additionally hindered by the lack of an integrated standard for IoT-based forensic investigation [4]. Existing methods rely on the data of interest being always readily accessible, which can not always be said for IoT related data.

The idea of using **blockchain technology** in digital forensics is becoming more and more interesting, as it provides a better alternative to the traditional centralized architecture, which is based on a central authority verifying the collected digital evidence. This provides an opportunity for the evidence being tampered with, either by insiders or attackers.

Blockchain is a list of blocks (records) that are linked together. Every block contains a cryptographic hash of the previous one, transaction data and a timestamp, which proves that the data existed when the block was published. The transaction data is usually represented as a **Merkle tree**, where each leaf is a data node. Blockchain is resistant to alterations, since no block can be modified without this impacting all blocks that follow it [1]. Using a blockchain for integrity preservation of data acquired in a forensic process is becoming more popular, as it makes the internal information and the ledger visible to all participants, therefore allowing any participant to verify data's integrity. There are, however, some constraints when it comes to using blockchain for IoT. The first is the constrained processing power and storage capacity of IoT devices which is necessary to secure and maintain a blockchain. Secondly, each new block's capacity is restricted (transactions are limited to a few dozen per second), which makes it hard to keep up with the rapid deployment of new IoT devices. Then there is the confirmation delay (single block creation), which would require a lightweight cryptographic hash algorithm [9], because of the aforementioned slow availability of new blocks.

The downside of traditional hashing techniques, which are usually used in blockchain, is the fact that every unique file/data stored on a blockchain, no matter how similar it is to some other data, will have a *completely different* transaction hash. This is due to its inherent property called *cryptographic diffusion* and it enables certain pieces of evidence (that might have similar binary strings as already documented evidence) to remain undiscovered. This is where **fuzzy hash** comes in. It enables comparing hashes of files that are *homologous*, which means they are similar, but are not perfect copies, since the similarity is retained in the hash. Using fuzzy hash, we can compare incomplete, malformed, partially obstructed files to the original in order to find the

connection with the suspect.

The method from the main article[5], which we aim to improve, has the difficulties of IoT forensics in mind, but any part of digital forensics could benefit from this architecture.

1.1 Article Overview

In chapter 2 we provide an overview of the proposed approach from [5], a brief description of its main components, and the authors' results in 2.5. In chapter 3, we describe the downsides of that approach and describe and propose our own architecture, which uses Smart Contracts on the Ethereum network. In chapter 4, we test and compare a few fuzzy hashing algorithms that can be employed within the two described architectures. The article is concluded with chapter 5, where we repeat our main findings.

2. PROPOSED APPROACH

The approach builds on previous work done on distributed ledger architectures, with the significant improvement of using **fuzzy hash** instead of *traditional hash* for preserving the integrity of blocks and easier analysis, as seen in figure 1. By combining these approaches it makes the forensic examination more manageable by accomplishing the following:

- Continual integrity for the whole evidence chain,
- Immutable storage of the record fingerprints,
- Transparency of the evidence chain,
- Identification of related records

In the following subsections, we will describe the main components of the framework and how the immutable chain of records fingerprints is constructed. We will refer to a hash of the record as a **fingerprint** or **digest**, since we are not excluding an option for the record to be digitally signed. Similarly, we will use terms such as **record** or (*transactional*) **evidence**, but they all refer to the same thing.

2.1 Proposed Forensic Chain

The proposed framework uses a distributed ledger for storing fingerprints of **transactional evidences** (TEs), while enabling users to sort related records based on **fuzzy hash**.

2.2 Transaction

Forensic chain starts by **fuzzy hashing** the data, producing a transaction hash that uniquely identifies it's corresponding data. Since each node on a blockchain should retain a copy of the complete ledger, it's reasonable to keep the amount of data stored on the chain as low as possible. To further optimize the amount of data stored on each block, a Merkle tree is constructed.

2.3 Merkle Tree

The **Merkle tree** is a hash tree, constructed by hashing all transaction hashes until it converges into a single hash called Merkle root, as seen on the Figure 2. Merkle root ensures that the each investigation's findings can be securely verified against it, requiring only reduced number of hashes that it relates to, rather than all hashes from the tree.

2.4 Blocks

To form an immutable object, Merkle root is combined with the following items to form a single block on the blockchain: (i) pre-block hash, (ii) which version it is, (iii) the nonce, (iv) the timestamp, (v) the block state and (vi) the Merkle root. Joining the described components we form an unbreakable chain of records, transparently stored on a decentralized network.

2.5 Results of Proposed Approach

Authors of [5] implemented the proposed model in **Python** and tested its response time, delay and throughput capabilities. They started by building the Merkle tree with fuzzy hash after applying SHA256 to the TE records, then validating the blockchain, using fuzzy hash for the Merkle root and then implementing the PoW and creating a text file with the block information.

First, they compared node response time in the cases of using fuzzy and traditional hash to create the Merkle tree. They measured the time the node takes to receive the transaction, mine a new block and create the text file with block information. The results show that in this model, using fuzzy hash reduces the response time by an average of 2% when compared to the version using traditional hash to encode transactional evidence records. This confirms that the model can deal with a large number of blocks and therefore validates the idea of using this model in real time.

They then tested the CPU usage in dependency of the number of generated blocks. As they expected, needed CPU resources increase with the need for new blocks, although the need for CPU resources seems to stagnate with the larger number of needed blocks. To stress-test, they simulated 50, 100 and 500 concurrent users with the users performing operations like getting the blockchain status, posting the transaction to the node and getting the mined block status. They observed that the completion time increased with the number of users and that getting the blockchain status and mining took the most time, which makes sense since mining needs more resources to complete PoW and hashing the Merkle root.

Lastly they tested the model on a Raspberry Pi, to confirm its ability to run on IoT devices. They considered the Raspberry Pi as a node in the blockchain network and focused on monitoring time consumption and power consumption, as these are the main issues running blockchain on IoT. They measured the power using a wall outlet power meter. The results were compared to consumption when using a laptop, as you can see in table 1.

	Laptop	Raspberry Pi
$\bar{t}$[s]	3	13
$\bar{P}$[mW]	4	12

Table 1: Average measurements of time and power consumption of generating a new block and adding it to the blockchain network. [5]

The time consumption difference makes sense since we are comparing a laptop with more resources compared to a Rasp-

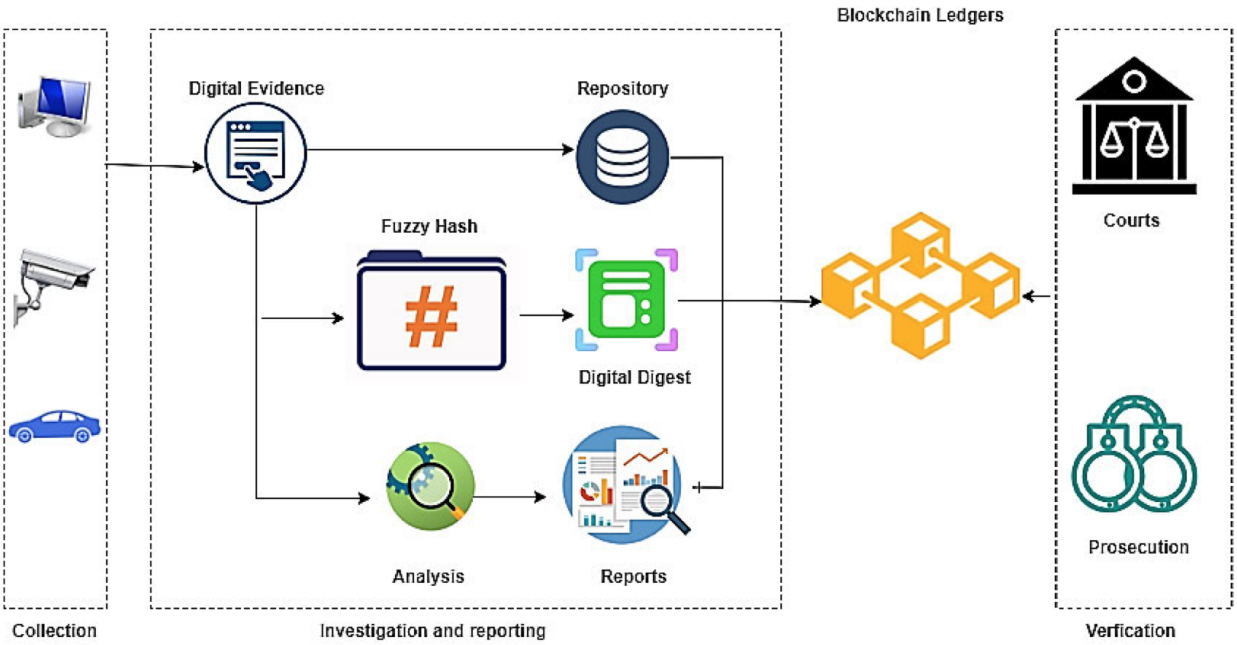

Figure 1: IoT evidence management using blockchain and fuzzy hash [5].

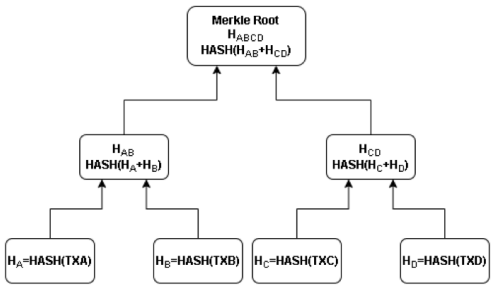

Figure 2: Merkle Tree

berry Pi. If we then compare this difference to the difference in power consumption, it is smaller.

The model does show some space for improvement on its execution time and complexity, which requires further research and testing. In any case, the proposed model seems like a good option even for implementations on devices with power and memory scarcity (like IoT devices).

3. ALTERNATIVE APPROACH

The downside of authors approach lies in the fact that in order to accomplish it, core algorithms of the blockchain need to be modified in order to suffice the need for similar records identification. We believe a better approach is to utilize Smart Contracts on Ethereum network, that enables users to build **decentralized applications** (DApps) on top of the blockchain, while preserving the underlying technology.

The following sections summarize the main components of the alternative approach and compare the two methods. Note that the two models mainly differ in terms of usability, while both address the same issue.

3.1 Data Digest

Each execution in the **Smart Contract** (SC) costs a fee, therefore it's desirable to perform as many as possible operations outside of the network. In other words, before interacting with the SC, we **(fuzzy) hash** the data locally - not to mention hashing is a costly operation and with this, we avoid higher costs. Each new hash of the potentially modified evidence is chained by prefixing it with the fuzzy hash of the evidences previous state which allows us to chronologically order the records.

Additionally, when pushing data to the blockchain it's advisable to lower the amount of it as much as possible. Not only would it take undesirable amount of time to complete the transaction (about 14 minutes to save just 1 MB of data), but it would also cost much more than to store the data just on an occasional database. In the following section we describe an additional technology used to store data and circumvent this issue.

3.2 Data Storage

In comparison to centralized database, **InterPlanetary File System** (IPFS) is a decentralized database we utilize to preserve transparency and immutable storage of the data. Primary purpose of this technology is not meant to store sensible information since any participant of the IPFS network can host and see your files. Therefore an additional public key infrastructure is needed, to encrypt the data, but this is out of scope of this article.

Stored data is content addressable enabling user to retrieve

the content based on an unique identifier. On top that, IPFS network allows us to reference arbitrary content using a single identifier which is achieved by **InterPlanetary Name System** (IPNS). The concept is summarized in Algorithm 1. In the subsequent chapters we will refer it as an IPNS identifier and use it to group evidence and its modification in the Smart Contract. In practice, the user is still required to keep a separate note of relevant content addresses.

Our proposed solution is not bound to use IPFS as a storage platform, rather the main component of the system is to use Smart Contracts to enable any party to self-verify the integrity of data.

Algorithm 1 Storing data on IPFS

INPUT: Digital evidence DE
OUTPUT: IPNS identifier I : Fuzzy hash F
$K \leftarrow Storedata\_on\_IPFS(DE)$
$I \leftarrow BindCID\_to\_IPNSidentifier(K)$
$F = calculate\_fuzzy\_hashed\_value(DE)$
return I, F;

3.3 Smart Contract

Smart contracts are digital contracts, which are kept on the Ethereum network and execute in a secure and immutable way while not requiring a third-party to communicate with the rest of the network.

When applied to IoT devices, **Proof of Work** (PoW) algorithm needed to establish a consensus between nodes about the ledger requires a lot of resources (memory and processing power), consequently a simplified consensus algorithm needs to be considered. Note that this issue is already partly addressed by the new Proof of Stack (PoS) consensus algorithm Ethereum community is actively developing.

With IoT, using a simplified PoW algorithm with a lower difficulty decreases the time needed to create a new block and with that, makes reaching the consensus between the nodes of the IoT network easier and much faster.

4. HASH ALGORITHMS COMPARISON

The proposed architectures both utilize **fuzzy hashing** instead of traditional hashing algorithms, to make sure that the integrity of the blockchain is maintained, and to enable preservation of similarity between files. This can not only detect whether a file has been tampered with but also provides a measure of change. In this chapter, we will compare 3 popular fuzzy hashing algorithms to see which one may be best suited for the purposes the architectures seek to achieve.

4.1 ssdeep

The first algorithm we will consider is `ssdeep` [3], probably the most popular fuzzy hashing algorithm. The process of `ssdeep` fuzzy hashing begins by splitting the file into several blocks, which are then each separately hashed. At last, all block hash values are then concatenated to create a joined fuzzy hash value. The length of the resulting fuzzy hash is influenced by the file size, block size and the output size of the chosen hashing algorithm. The proposed method in [5] precisely uses `ssdeep`, which outputs context-sensitive piecewise hashes [2] (**CTPH**), which is also what fuzzy hashes are called. This means we are able to identify identical bytes in the same order, even if bytes separating them vary in length and content. Files like that are also called **homologous**.

After the fuzzy hash is constructed, the digital investigator can make conclusion on the size of modifications. He is also able to determine which block of the files were changed. `ssdeep` outputs a measure of similarity between two files, meaning a 100 denotes files are equal, while 0 means files are totally different.

4.2 LZJD

The next fuzzy hashing algorithm we will have a look at is `LZJD`, or **Lempel-Ziv Jaccard Distance** [7, 8]. `LZJD` is a distance metric designed for arbitrary byte sequences, and was originally used for malware classification. The background and motivation for `LZJD` was given in [8], but here we're going to briefly cover what `LZJD` does to compute file similarity.

This method works by building a set of previously seen strings from the given byte string b. The set initially starts out empty, and a pointer starts at the beginning of the file looking for substrings of length 1. If the pointer is looking at a substring that has been seen before, we leave it in place and increment the desired substring length by 1. If the pointer is at a substring that has not been seen before, it is added to the set. Then the pointer is moved to the next position after the substring, and the desired substring length reset to 1. This is repeated until no new items can be added to the set. The function returns the constructed set. The strings in the set will get progressively longer as the length of the input increases. A pseudo code, for easier understanding, is given in Algorithm 2.

Algorithm 2 Lempel-Ziv set [8]

```
1: Input: Byte string b
2: S ← {}
3: start ← 0
4: end ← 1
5: while end ≤ |b| do
6:     b_s ← b[start : end]
7:     if b_s ∉ S then
8:         S ← S ∪ {b_s}
9:         start ← end
10:    end if
11:    end ← end + 1
12: end while
13: return s
```

Once we have these sets for two binary strings of interest, we can compute the **Jaccard similarity** between these two sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The value $J(A, B)$ is in the range $[0, 1]$, so the output of `LZJD` is a percentage which indicates how similar two byte strings are. Computing this similarity, especially for longer byte strings, turns out to be rather slow, so approximations are used to speed up the process. However, we will not go into details here, as this is not the topic of this paper.

(a) Original image with no watermark (b) Changed image with added watermark (bottom right)

Figure 3: Images used for comparing fuzzy hashes

4.3 TLSH

The last fuzzy hashing algorithm we will consider here is `TLSH` or **Trend Micro Locality Sensitive Hash** [6]. Given a byte stream with a minimum length of 50 bytes, `TLSH` will generate a hash value which can be used for similarity comparisons. We will shortly cover the steps `TLSH` does to produce a hash value for a byte string.

Observe any byte string. The `TLSH` hash of that byte string is computed during 4 steps:

1. Process the byte string using a sliding window of size 5 to populate an array of bucket counts.
2. Calculate the quartile points of the bucket array, q_1, q_2 and q_3
3. Construct the digest header values (header of the digest has 3 bytes)
4. Construct the digest body by processing the bucket array

The final `TLSH` digest constructed from the byte string is the concatenation of the hexadecimal representation of the digest header values from step 3, and the hexadecimal representation of the binary string from step 4. The scoring function that is used for `TLSH` is rather complicated, but it is just important to point out that in theory, it is in the range $[0, \infty]$. i.e. it is theoretically unbounded from above, unlike `ssdeep` and `LZJD`. This means the higher the `TLSH` score, the more differences exist between the two binary strings we are comparing (while a score of 0 of course means the two strings are identical). Likewise, in the original paper for `TLSH`, the authors reason the choices for some of the sub-parts of each step, but we will not go into that here. We will immediately dive into comparing these 3 fuzzy hashes.

4.4 Comparison

Let us now perform the comparisons. First off, consider two pieces of text, with only one change being that the second text uses a capital B for the first occurrence of the word "building":

- "Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: ***building*** blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks"
- "Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: ***Building*** blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks"

The small Python script we wrote returned the result seen in figure 4.

```
TEXT COMPARISON:
TLSH hash comparison(the lower the number, the better):
difference = 84

SSDeep hash comparison(the higher the number the better):
similarity = 97

LZJD hash comparison(the higher the number the better):
similarity = 0
```

Figure 4: The comparison/similarities returned for the provided text

We see in figure 4 that `ssdeep` seems to be the best one out of the bunch. It returns that the two short paragraphs are 97% similar. On the other hand, `LZJD` was not able to detect any differences between them, while `TLSH` detected it. If we make another change, and instead of the word being "Building", we set it to "BUilding", `ssdeep` now returns a similarity of 100%, while `TLSH` returns the result of 87 (only an increase of 3 in the difference score). `LZJD` still returns 0. If we set the word to be "BUIlding", `ssdeep` returns

97%, while TLSH surprisingly returns only 11. Even though the second result returned by `ssdeep` is not the most accurate, we can definitely turn our attention away from `TLSH`, as it showed major inconsistencies with the returned score. Hence, `ssdeep` seems to be (way) more consistent with its results (at least for text).

Now consider two pictures, one without a watermark (figure 3a), and the other one completely the same, but with our watermark (figure 3b) - this picture was altered ever so slightly. A change like this, in practice, may indicate that the criminal tried to portray the image as his own, i.e. he denies any copyright infringement or plagiarism. The results returned by our short script are shown in figure 5.

```
IMAGE COMPARISON:
TLSH hash comparison(the lower the number, the better):
difference = 199

SSDeep hash comparison(the higher the number the better):
similarity = 0

LZJD hash comparison(the higher the number the better):
similarity = 0
```

Figure 5: The comparison/similarities returned for the shown images

For reference, if we take any picture, change one byte of it, and compare those two pictures, `TLSH` will return a result of just 1! (while `ssdeep` and `LZJD` both return 0). Now, even though the pictures are slightly different, we notice (in figure 5) that `ssdeep` and `LZJD` fail to detect the differences, while `TLSH` detects it, with a difference of 199. It is a noticeably larger score than 1, however, unlike `ssdeep` and `LZJD`, it is at least sensitive enough to detect the differences. The similar result is obtained with other, similar images (one original, and one with a very light watermark). Out of these three algorithms, we can say that `TLSH` is the most consistent for images.

5. CONCLUSIONS

The approach outlined in [5] is a rather adequate approach preservation of data integrity. Above all, it allows digital investigators to carry out their investigations and examinations in a more manageable and modern way, while still providing more or less completely secure evidence preservation. A blockchain based digital forensic framework is a perfect for this, but the described approach does not come without downsides. The problem lies in the fact that in order to successfully accomplish it, the core of the blockchain needs to be changed, only such that similar records can be identified.

Our approach utilizes Smart Contracts running on the Ethereum network. This enables users to build decentralized applications on top of the blockchain, while preserving the underlying technology. This removes the need to change the core of the blockchain, while solving the exact same issue as the original approach. Likewise, our architecture enables users to use also other fuzzy hashing technologies interchangeably, making it backward compatible and easily upgradeable in the future.

Setting aside the resources required to secure and efficiently run a larger scale blockchain network for IoT devices, the lightweight and popular fuzzy hashing algorithms that would be algorithms of choice, namely `ssdeep` for text based evidence, and `TLSH` for images and videos as discussed in section 4, are easy to use and the code is open source too. Unfortunately, we are yet to see a fuzzy hashing algorithm that can work well for both images *and* text at the same time, but our proposed architecture is constructed in such a way where any fuzzy hashing algorithm can be used.

Using blockchain for integrity preservation of evidence during a forensic investigation is growing in popularity, especially in recent years. The difficulty of preserving the integrity of our evidence (or anything for that matter) is built right into and literally defines blockchain as a concept, and we are making use of exactly that. The aforementioned lightweight fuzzy hashing algorithms are provably good for digital forensics purposes. Regardless of the drawbacks of the approach described in [5], a blockchain-based digital forensics architecture brings many advantages over traditional investigation and evidence storing techniques, and as this technology develops, it is only going to become easier for digital investigators to perform their already cumbersome investigations.

6. REFERENCES

[1] Cao, Bin and Liu, Weikang and Peng, Mugen. *Blockchain-Driven Internet of Things*, pages 93–115. 2022.

[2] J. Kornblum. Identifying almost identical files using context triggered piecewise hashing. 3, 2006.

[3] Kornblum, Jesse. Identifying Almost Identical Files using Context Triggered Piecewise Hashing. Digital Investigation 3(suppl.), 91-97. *Digital Investigation*, 3:91–97, 09 2006.

[4] MacDermott, Áine and Baker, Thar and Shi, Qi. Iot Forensics: Challenges for the Ioa Era. pages 1–5, 02 2018.

[5] Mahrous, Wael A. and Farouk, Mahmoud and Darwish, Saad M. An Enhanced Blockchain-Based IoT Digital Forensics Architecture Using Fuzzy Hash. *IEEE Access*, 9:151327–151336, 2021.

[6] Oliver, Jonathan and Cheng, Chun and Chen, Yanggui. TLSH – A Locality Sensitive Hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, pages 7–13, 2013.

[7] Raff, Edward and Nicholas, Charles. An Alternative to NCD for Large Sequences, Lempel-Ziv Jaccard Distance. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1007–1015, New York, NY, USA, 2017. ACM.

[8] Raff, Edward and Nicholas, Charles K. Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash. *Digital Investigation*, feb 2018.

[9] Tomoyasu Suzaki and Kazuhiko Minematsu and Sumio Morioka and Eita Kobayashi. TWINE : A Lightweight Block Cipher for Multiple Platforms. In *Selected Areas in Cryptography*, 2012.