Efficient Cyber-Evidence Sharing using Zero-Knowledge Proofs*

Arman Zand[1] and Eckhard Pfluegel[1]

Kingston University, Kingston upon Thames, United Kingdom
{a.zand,e.pfluegel}@kingston.ac.uk

Abstract. A common challenge for organisations managing confidential customer information is to respect obligations due to legal requirements while advocating the privacy of their users' data. In the context of digital forensics and cyber security scenarios, we define cyber evidence sharing as the task of verifying that mutual knowledge exists about confidential information or digital evidence, without revealing or disclosing the information itself. An attractive cryptographic solution to this problem is the concept of Zero-Knowledge Proofs (ZKPs). In this paper, we propose a flexible and efficient approach for sharing cyber evidence based on using ZKPs. We present a protocol that allows a verifier to establish the proof of knowledge of hash digest information. This provides computational security and can be implemented efficiently using a Merkle-tree data structure. The protocol has been implemented, the resulting proof-of-concept system is evaluated, and its efficiency is demonstrated. We believe that it could be a valuable tool for use in practical real-world applications.

Keywords: Privacy · Cyber Evidence · Information sharing · Zero-Knowledge Proofs.

1 Introduction

Data is a critical asset for organisations and is under constant threat from disclosure, modification or destruction by external or internal attackers. Whether it is in the healthcare, finance or energy sector, there are many different types of sensitive information being managed and maintained: human behaviour, client credentials or intellectual property, biometric data or any other personal information. A particular sensitive data asset is information about suspected illegal, criminal activities that an organisation might have of their customers. While this information is recorded and protected within specific departments, they may not be able to disclose it to other parties, even within the same organisation, for legal reasons. On the other hand, if several departments independently acquire this information, there might be a need to exchange this knowledge and to take further action. Sharing the information must be done in such a way

* This research has been partially supported by the Kingston University Faculty of Science, Engineering and Computing.

that it maximises privacy. In particular, no unnecessary information about the nature of the allegations must be revealed. There are various legal obligations that organisations have to adhere to, concerning their collection, use, storing and sharing of data. It is the latter aspect that is of interest in this paper, in the context of cyber evidence sharing. For example, a regulatory body may interpret sharing of information as breaking anti-trust laws, due to sensitive competitive information involved in cyber evidence sharing. There is also a risk in sharing sensitive information that is liable to violating laws that protect privacy. It may also be difficult for a company to decide on sharing threat intelligence with the government for which the regulators can choose to punish the company for unsatisfactory security practices. For example, Uber was fined $148 million in 2018 after paying $100,000 for non-disclosure to a perpetrator that found data exploits in 2016 [4].

We define cyber evidence sharing as the task of verifying that mutual knowledge exists about confidential information or digital evidence, without revealing or disclosing the information itself. Cyber evidence sharing constitutes a significant challenge, and traditional solutions require the involvement of a trusted third party as an intermediary, for example, the police or an ombudsman. However, there is always a risk of information leakage, as trust can never be completely guaranteed. For this reason, in this paper we propose the use of cryptography, thereby relying on openly published security solutions, in order to tackle this challenge. An attractive scheme that can be used as a solution to this problem is the concept of Zero-Knowledge Proofs (ZKPs). A ZKP is an asymmetric protocol, run between a *prover* Peggy (who will be represented by Department P) as well as a *verifier* Victor, aka Department V. In our scenario, we will assume that cyber evidence information is presented as structured data D and that a hash digest value $H = h(D)$ has been created, using a secure hash function h. Department P and Department V wish to prove the correctness of the mutual knowledge of H. ZKP properties ensure that the protocol is *complete*, ensuring that in any scenario with a piece of correct shared knowledge, it can be verified by Department V. Furthermore, the protocol is *sound*, meaning that Department P cannot trick Department V into proving wrong information. The *Zero-knowledge property* implies that no additional information other than the outcome of the proof can be obtained.

ZKPs have been invented in the 80s [3] and subsequently extended and refined for increasingly sophisticated types of proofs. Since only fairly recently, there has been a considerable uptake of ZKPs and their use in innovative applications such as the Blockchain. For example, ZK-SNARKs [6] allows verifying the correctness of a computation, rather than just a value. This scheme is derived from decades of research of Microsoft's Pinocchio Protocol [9] which made the protocol succinct, non-interactive and zero-knowledge. ZK-SNARK is a complex protocol due to the series of components required to flatten code into verifiable proofs. The result however is a powerful scheme, which enables cryptocurrencies to provide privacy by having encrypted transactions being verifiable using ZK-SNARK proofs. The

aim of our research is to add to the list of innovative applications of ZKPs by looking at its use in cyber evidence signalling.

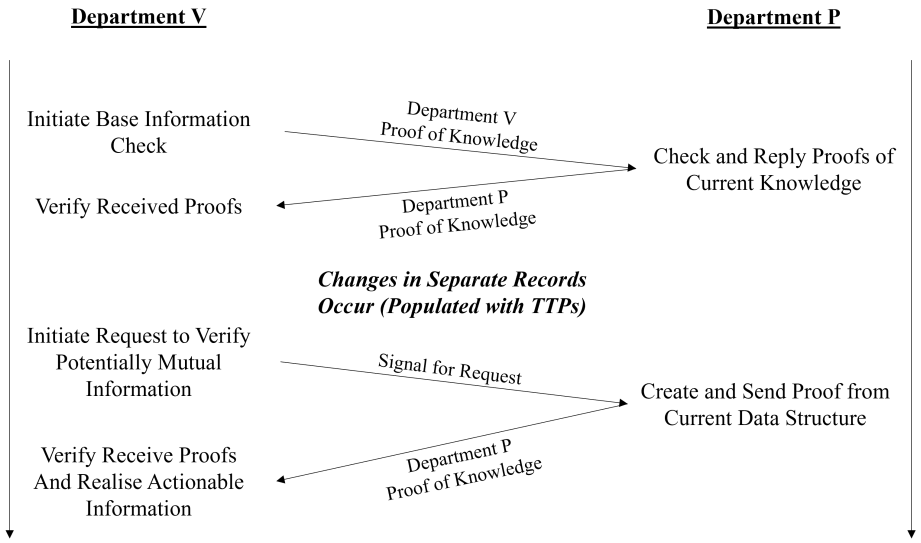

Fig. 1. A brief overview of our proposed solution to Cyber-Evidence Sharing. Department V and P begin by establishing that they both exactly have the same base information. With both parties having the same base data, they proceed to begin populating these records with Tactics, Techniques and Procedures (TTPs, [7]). After some time, a department may send a signal to the other department to request verification of any mutual information that has been discovered since populating the records. The signal is replied with proofs of knowledge derived from the current department's data structure. The initiator now verifies these proofs using their own knowledge and are then able to realise any actionable information.

The contribution of this paper is the design, implementation and evaluation of a proof-of-concept system for simple and yet flexible as well as efficient verification of knowledge in the context of cyber evidence sharing. The communication scheme that is used builds on sound cryptographic schemes [3] and establishes a non-interactive protocol between the departments, wishing to verify knowledge without leaking any additional information. Rather than designing a complex scheme, we present a simple protocol that allows a verifier to establish the proof of knowledge of hash digest values, obtained from structured information in any format. This offers flexibility and would be suitable for e.g. the use of frameworks such as Tactics, Techniques and Procedures (TTPs, [7]) which allows the forming of a coherent and detailed description of cyber attacks. A careful choice of data structures using a Merkle-tree improves the efficiency, when dealing with repeated protocol runs required for realistic scenarios involving multiple report

verification. We demonstrate the achieved efficacy by comparing argumentation to a naive implementation, using a linear list. We believe that our tool could become a valuable tool for use in practical real-world applications. This paper is structured as follows: in Section 2, we outline past work comprising of both the aspects of cyber evidence sharing and practical ZKP schemes. Section 3 contains a description of our protocol design, followed by its implementation and evaluation in the next section. Section 5 outlines the more complex alternative to the ZKP method used in the our scheme and finally the paper is concluded in Section 6.

2 Previous Work

In this section, we discuss previous work on traditional mechanisms for cyber evidence sharing as well as the Non-Interactive Fiat-Shamir Identity Scheme which is a specific ZKP that we have used in our system.

2.1 Traditional Cyber Evidence Sharing Approaches

There are several solutions that attempt to share cyber evidence between parties legally, although these do not take advantage of any means that are cryptographic in nature. These include LISTSERV, Information Sharing and Analysis Centers (ISACs), Information Sharing and Analysis Organizations (ISAOs) [10] which aim for a collaboration on sharing threat intelligence. Additionally, subject to any sovereign nation, there are public sector national cyber centres to work with [7]. These are however constrained by lengthy and bureaucratic processes for which a cryptographic solution would give advantages such as the removal of an operating trusted third party. Although the scope of the paper is not to exhaustively explore these non-cryptographic solutions, there is effectively no extensive previous work on the topic that uses cryptography to solve this problem. In addition, it is difficult to find solutions for such legal and confidential matter for which organisations will have built their own private solutions and models.

2.2 The Fiat-Shamir ZKP Scheme

It is critical to review the concise description of the Fiat-Shamir Non-Interactive Identification Scheme [13] to understand the interactions in the application and its context in the evaluation. The Scheme may also be described as Random Oracle Access which is a ZKP Scheme used to verify that there exists common knowledge and is not probabilistic. This is a non-interactive protocol of the Feige-Fiat-Shamir Identity Scheme [2] which uses the Fiat-Shamir Heuristic to prove the correctness of proofs [3]. We have chosen this particular scheme because it fundamentally achieves the objective of verifying mutual information without disclosure in a way that constructing this proof system is not complex.

The non-interactive version facilitates its implementation in both the cryptographic method and network messaging where less messages are required to be exchanged. The scheme itself is not complex to implement because computing the proof is a matter of modular exponentiation which can be suited for microprocessor devices. The method of checking the correctness in the verification relies on the exponentiation of; $r = v - cx$ which is also present in Schnorr's Signature Scheme [12]. In addition, the scheme will be used multiple times to check multiple secrets and there is an opportunity to merge all the secrets into a single value.

Table 1. Table of notations and definitions for a Non-Interactive Fiat-Shamir Identity Scheme.

Notation	Definition
p	Prime integer.
g	Generator value.
x	Alleged common value.
v	Random value.
t, c, r	Commitment, Challenge & Response respectively.

The scheme takes two entities, Peggy and Victor, Prover and Verifier respectively for which they share g, p and will only successfully complete the protocol if x is the same. To begin, both parties compute y with their independent knowledge of x:

$$y = g^x \bmod p \tag{1}$$

CreateProof() : Peggy must compute the following pair of values (r, c) which is the proof to be sent to Victor. Peggy chooses a random $v \in \mathbb{F}_p$ and computes the following:

$$\begin{aligned} t &= g^v \bmod p \\ c &= \text{Hash}(g, y, t) \\ r &= v - cx \bmod p \end{aligned} \tag{2}$$

Verify() : Victor now works with the proof (r, c) that he receives from Peggy and also y derived from his own knowledge of x that is supposed to be the value that Peggy is attempting to prove. Victor will reconstruct t by using r, c from Peggy's proof and subsequently allows Victor to compute $\hat{c}$ and compare it with Peggy's c. Victor can reconstruct t without knowing v because he uses $g^r = g^{v-cx}$ and $y^c = (g^x)^c \therefore g^{v-cx} \times g^{cx} = g^{v-cx+cx} = g^v = t$.

$$\begin{aligned} \hat{t} &= g^r \times y^c \pmod p \\ \hat{c} &= \text{Hash}(g, y, \hat{t}) \\ (\hat{c} = c) &\rightarrow \textbf{Verified} \end{aligned} \tag{3}$$

3 Design

In this section, the design of the proposed system is discussed featuring the data structures and its protocol.

3.1 Merkle-Trees

Merkle Trees are generally used as an efficient representation of a data structure to be used for comparison. Assuming that most of the data blocks are the same, the efficiency is $O(\log_2(n))$, for n depth of the binary tree. A popular use case is cryptocurrency transaction blocks being represented as Merkle Trees. Cryptographic hash function such as SHA256 or higher are used in Merkle Trees where each conjunction of 2 nodes is concatenated and hashed until a single root node is left. The Merkle Tree allows a fast comparison of some blocks of data. For each leaf node that doesn't match, participants can enter a sub protocol to discover further mutual knowledge in that subset data block. For this, further subset of data in the block is converted into a list of prompts for other participants to prove in the form of ZKP proofs. The participants will agree on a list of queries and the format for which they are answered where each query prompts for potential mutual knowledge. Queries will be answered as ZKP Proofs for which the query creator will verify them. From this exchange, it is possible for participants to acknowledge mutual information and proceed on actionable information.

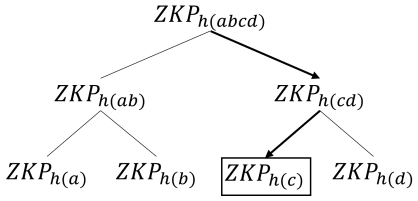

Fig. 2. ZKP Binary Tree where each node is a Proof derived from the hash digests of the original Merkle Tree. The bold arrows show the path taken down the tree where there is a difference in information. The hash comparison function is replaced with ZKP Verification which prevents participants from learning any new information.

3.2 ZKP

By using the scheme outlined in Section 2.2, the functions are arranged to take binary trees as inputs where subsequently each node can be operated on. Therefore, a CreateProof function taking a Merkle Tree will go through each node's hashes and convert them into proofs. The proofs retain the structure of the binary tree given. Likewise a Verify() function will take the output binary tree of the CreateProof() and also a normal Merkle Tree to compare with where the output results in a list of unverifiable leaf nodes.

3.3 Protocol

Our network consists of a centralised server relaying messages between participants. These communication channels are secured conventionally using certificates and TLS. The protocol we outline using Fiat-Shamir Identity scheme does not provide non-repudiating authentication. Our proposed protocol is therefore simplified and safer by having the setup of communications being delegated to TLS.

Department V and Department P may interchangeably become provers or verifiers in different parts of the protocol. For the example in Figure 3, Department P is the prover throughout, although this role may be swapped in the intermediate stage where separate changes in records occur. In terms, of the data structure, it is possible to include each record and their subset elements as one Merkle Tree, however the protocol respects the original data structure hierarchy and stops before going into the next dimension.

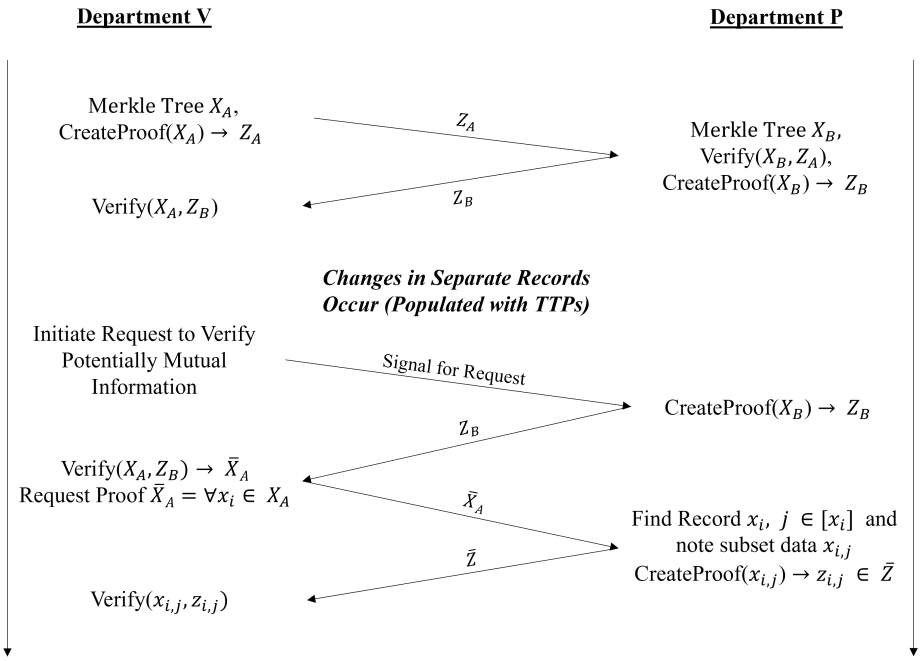

Fig. 3. A more detailed overview of our protocol which describes the use of ZKP in combination of Binary Trees. See Protocol 3.3 for the protocol steps.

Protocol 1 Cyber-Evidence Sharing

Goal. Parties compare large lists of data and establish mutual knowledge without revealing or learning any new information.
The protocol:

1. Each party begins with the same base knowledge represented in a Merkle Tree X. They then check if they have the same data by converting their Merkle Tree nodes into proofs using the proof creation method in Section 2.2. That is, CreateProof$(X) \rightarrow Z$ for which Z is exchanged and verified against their own X in Verify(X, Z).
2. As time passes, parties populate their own records with their discoveries of TTPs.
3. A party can initiate a check to verify any mutual information where other parties become provers. The party sends a signal to request some proof, parties that receive the signal compute CreateProof$(X) \rightarrow Z$ where X is the binary tree representation of the current private knowledge that includes any recorded TTPs. This proof is replied back and verified by signal sender with Verify$(X, Z) \rightarrow \bar{X}$. The result $\bar{X}$ is a list of unverified nodes $x_i \in X, i \in [X]$.
4. Request proof for the subset information of all unverified nodes $x_i \in X$. The party fulfilling this request finds record x_i which contains j elements, thus $j \in [x_i]$. A proof is derived from this by computing CreateProof$(x_{i,j}) \rightarrow z_{i,j} \in \bar{Z}$ where $z_{i,j}$ is the proof representing element $x_{i,j}$. This is sent back to the requesting party.
5. The received list of proofs $\bar{Z}$ is used to verify any potential mutual information in a record by checking subset data Verify$(x_{i,j}, z_{i,j})$.

4 System Implementation and Evaluation

In this section we report on the implementation of our system, and its evaluation comparing the performance of proof creation and verification using the method outlined in Section 2.2 with a variable number of secrets.

4.1 Implementation

The system has been implemented on a Windows 10 machine using an i-9 10900K CPU @ 3.70 GHz (up to 5.30 GHz) on a single thread where the implementation of the ZKP protocol is written in C#. The implementation of the Merkle Tree in Section 3.1 and the scheme in Section 2.2, was done using standard libraries on the .Net Framework 4.7.2. Namely, the BigInteger library was used to handle arbitrary precision integers and SHA256 was used as the hashing function and converting the secret into an integer. This also includes the secure cryptographic pseudo-random number generator provided by the standard library.

4.2 Fiat-Shamir Identity Scheme Performance

In this section, we evaluate the Non-Interactive Fiat-Shamir Identity Scheme in itself for the time taken in proof creation and proof verification. Figure 4 shows using a box plot that a single proof creation for a 20 byte input is fast (in the order of milliseconds). Proof verification is only slightly slower, and still using the same method, the functions' performance is linear whilst still in the range of milliseconds. As shown in Figure 5, a 1 megabyte input takes around 12 milliseconds.

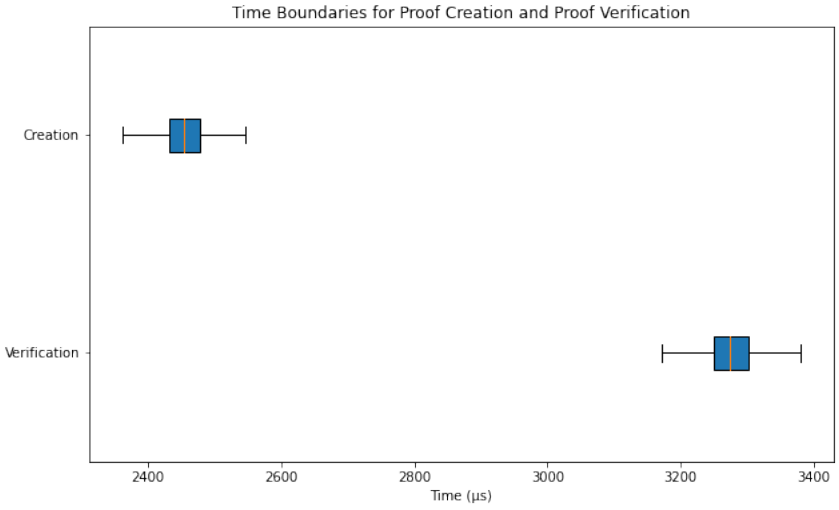

Fig. 4. Time boundaries for Creating Proofs and Verification of Proofs with secrets the size of 20 bytes. Time taken to compute a proof takes around 2.4 - 2.5 milliseconds whilst proof verification takes around 3.3 milliseconds.

4.3 Merkle Tree Performance

We now proceed to look at binary tree performance incorporated in our implementation which uses the ZKP method for verifying nodes. Evidently, the total computation time to produce a number of proofs is a multiplied with the time taken for one proof as previously discussed and is therefore linear. For large number of secrets, e.g. for 100,000 secrets, the time taken to convert the Merkle Tree hashes into proofs would take just over 8 minutes on a single thread. Although this takes considerably longer than creating the hash tree or verifying the proofs,

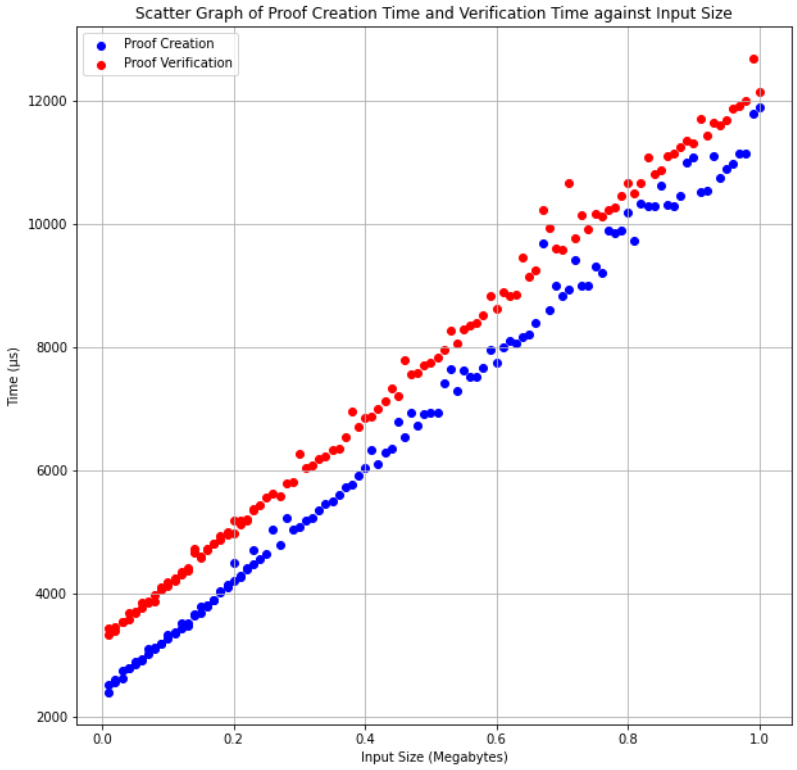

Fig. 5. Scatter Graph showing the time it takes to create and verify a proof for large secrets ranging from 10 kilobytes to 1 megabyte.

this process is improvable by several magnitudes with multi-threading. For periodic Merkle Tree transformations, the process is viable for some large lists although it may take a few minutes. The Binary Tree verification is compared with a linear list to show that indeed the verification process is logarithmic in Figure 6. The linear list procedure, Algorithm 1, is a single loop through each element. Meanwhile Algorithm 2, is a recursive procedure that verifies nodes from the root to the leaf nodes. It can be invoked using the following instructions:

if rootNodeZ.Verify(rootNodeM) = **False then**
Call RecVerify(rootNodeZ, rootNodeM)
end if

It is also worth to note that if the algorithm had to check for multiple differences at once, going through different paths will result in a longer total verification albeit logarithmic with respect to the number of nodes.

Algorithm 1 Verify all elements in linear list of proofs Z and list of local secrets M where $[N] = [M]$.

for $(i = 0; i < [Z]; i++)$ **do**
  Verify(Z_i, M_i);
end for

Algorithm 2 Verify all elements in a Binary Tree of proofs Z by comparing with a normal Merkle Tree of the same structure M. Each node contains a left and right child, if not it is a leaf node. Each node contains information that can be used in the ZKP verification function.

procedure RecVerify(NodeZ, NodeM)
  if NodeZ.leftChild.Verify(NodeM.leftChild) = **False then**
    RecVerify(NodeZ.leftChild, NodeM.leftChild)
  else if NodeZ.rightChild.Verify(NodeM.rightChild) = **False then**
    RecVerify(NodeZ.rightChild, NodeM.rightChild)
  end if
end procedure

4.4 Subset Data Verification

In this section, we consider an extension of the previously explained features of our protocol. We are interested in a situation where a modified record contains enough fields that warrant a further identification differences. For each record that is modified, there may be some subset of elements that are different. This task is essential done using either a Merkle Tree or Linear List depending on the

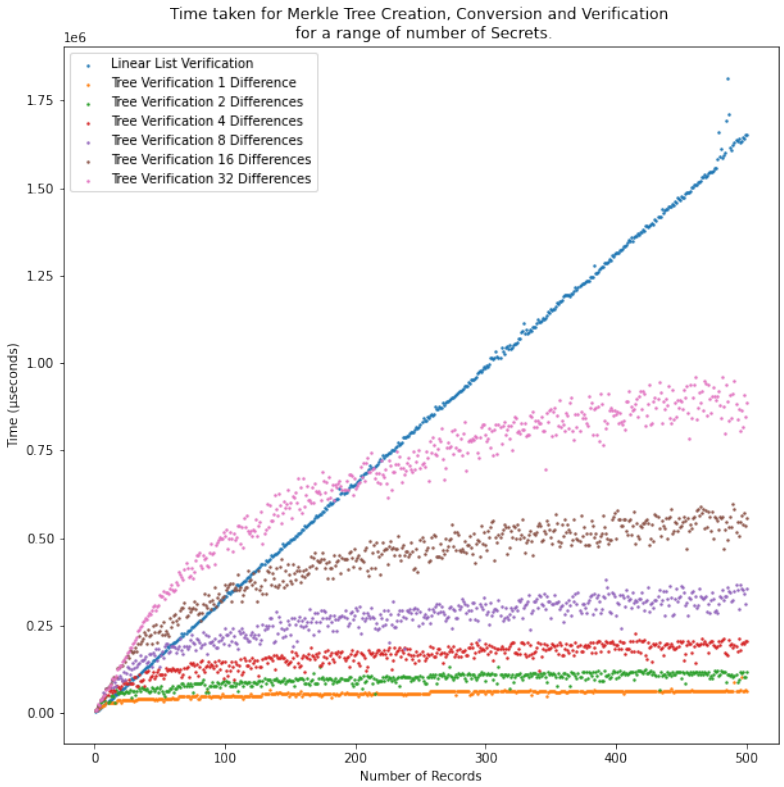

Fig. 6. Time taken to verify data structures containing Zero-Knowledge Proofs organised in a linear list compared against a binary tree of variable differences in leaf nodes. For one difference in the records, the latter is faster when there are 7 or more records. On the other hand, if there are a greater number of differences, e.g. 32; the Linear List is faster than the Merkle Tree for about 220 records.

.

size of this list. For example, each record contains 5 elements where in this case a linear list is sufficient however, if more than a dozen elements exist, using a Binary Tree for this task might become more attractive.

5 Related Work

In this section we further detail properties and current uses of the ZK-SNARK protocol, and we also discuss its potential use for our scenario.

As already stated earlier in this paper, Microsoft's Pinocchio Protocol [9], was taken and improved by the privacy orientated cryptocurrency community which resulted in the ZK-SNARK protocol [6]. This protocol features the following prominent characteristics:

- Zero-Knowledge – able to prove to another party a statement is true without revealing said information.
- Succinct – Must be verifiable in milliseconds and proofs produced are of short length.
- Non-Interactive – Protocol doesn't require back-and-forth communication between the prover and verifier.

The ZK-SNARK protocol is used as a method verify the authenticity of confidential transactions. Monero [11] and Zcash [1] have incorporated this scheme. In addition, a more popular cryptocurrency, Ethereum [8] has a research group which has explored ZK-SNARK to enhance privacy in transactions.

ZK-SNARK works by combining several concepts. Roughly speaking, code is flattened into gates, then converted to R1CS (Rank 1 Constraint System) [5]. This is now in a suitable format to convert into a QAP (Quadratic Arithmetic Program). As a QAP, it is possible to perform homomorphic hiding with blind polynomial evaluation. Furthermore, the security of the polynomials is enhanced by using Elliptic Curve Mapping. The improvements made in the succinct version [6] allow this scheme to be used by cryptocurrencies due to the short length proofs being produced.

While the ZK-SNARK protocol is very powerful, it is complex and requires specialist knowledge in order to be implemented correctly. For the purposes of our cyber evidence sharing scenario, we believe that it would be difficult to do such an implementation task without a major effort. We hence argue that our system and lightweight protocol is a reasonable alternative, ready for implementation in a real world organisation.

6 Conclusion

In this paper, we have used the ZKP Non-Interactive Fiat-Shamir Identity Scheme, to implement an approach for cyber evidence sharing within an organisation. The scheme guarantees that the successful verification of identical shared knowledge cannot be compromised and that no additional information

is revealed in the case of differing evidence data. The protocol that we have designed is simple yet flexible and efficient, due to the use of a suitable cryptographic data structure. This makes it an attractive solution for use in real-world scenarios.

One of the limitations of our system is the need for a semi-trusted environment, as the communication channels between the proving and verifying departments are potentially vulnerable to eavesdropping by any overarching entity at a higher level within the organisational hierarchy. Furthermore, providing anonymity could be an additional important security goal to implement.

As an item of further work, we remark that an interesting improvement of our cyber evidence sharing system would be an enhanced cryptographic ZKP protocol, addressing the proving of mutual code execution, without revealing the corresponding code akin ZK-SNARK. This would add another dimension to our proposed scheme and could be beneficial in a range of aspects. Apart from integrity, in the absence of malicious software alteration, it could also be applied to specific operations or actions that are triggered by the individual departments as a secondary investigation, following an initial successful cyber evidence sharing activity.

References

1. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE Symposium on Security & Privacy (Oakland). pp. 459–474. IEEE (2014)
2. Feige, U., Fiat, A., Shamir, A.: Zero-knowledge proofs of identity. Journal of Cryptology **1**(2), 77–94 (Jun 1988)
3. Fiat, A., Shamir, A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Advances in Cryptology — CRYPTO' 86, vol. 263 LNCS, pp. 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)
4. General, N.Y.S.A.: A.G. Underwood Announces Record $148 Million Settlement With Uber Over 2016 Data Breach, `https://ag.ny.gov/press-release/2018/ag-underwood-announces-record-148-million-settlement-uber-over-2016-data-breach`
5. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 626–645. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
6. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
7. Johnson, C., Badger, M., Waltermire, D., Snyder, J., Skorupka, C.: Guide to Cyber Threat Information Sharing. In: COMPUTER SECURITY. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD (2016-10-04 2016). https://doi.org/https://doi.org/10.6028/NIST.SP.800-150
8. Kirejczyk, M., Szlachciak, P., Jelski, K., Maretskyi, D., Wiech, A., Charczuk, J., Kirejczyk, N.: Ethworks Report: Zero-Knowledge Blockchain Scalability (2020), `https://ethworks.io/assets/download/zero-knowledge-blockchain-scaling-ethworks.pdf`
9. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252 (2013). https://doi.org/10.1109/SP.2013.47
10. Peretti, Kim: Legal Issues In Sharing Cyber Threat Intelligence: What Are The Real Concerns? 2015 Cybersecurity Innovation Forum – Alston & Bird (9 2015), `https://csrc.nist.gov/CSRC/media/Presentations/Legal-Issues-in-Sharing-Cyber-Threat-Intellige-(1)/images-media/day1_info-sharing_1100-1150.pdf`
11. van Saberhagen, N.: Cryptonote v 2.0 (2013), `https://bytecoin.org/old/whitepaper.pdf`
12. Schnorr, C.: Efficient identification and signatures for smart cards. In: CRYPTO (1989)
13. Stadler, M., Markus, J.C.: Proof Systems for General Statements about Discrete Logarithms. Tech. Rep. 260, Dept. of Computer Science, ETH Zurich (1997). https://doi.org/10.1.1.56.1208