Contents lists available at ScienceDirect

Journal of Information Security and Applications

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

Shared file protection against unauthorised encryption using a Buffer-Based Signature Verification Method

Arash Mahboubi [a,*], Seyit Camtepe [b], Keyvan Ansari [c], Marcin Pawłowski [d], Paweł Morawiecki [e], Hamed Aboutorab [a], Josef Pieprzyk [b], Jarek Duda [d]

[a] *Charles Sturt University, Australia*
[b] *Data61, CSIRO, Australia*
[c] *Murdoch University, Australia*
[d] *Jagiellonian University, Poland*
[e] *Polish Academy of Sciences, Poland*

ARTICLE INFO

Keywords:
Ransomware
Data encryption
Signature embedding
Storage-level signature validation
Trusted Platform Module
Coloured Petri net
Data protection

ABSTRACT

Understanding the attributes of critical data and implementing suitable security measures help organisations bolster their data-protection strategies and diminish the potential impacts of ransomware incidents. Unauthorised extraction and acquisition of data are the principal objectives of most cyber invasions. We underscore the severity of this issue using a recent attack by the Clop ransomware group, which exploited the MOVEit Transfer vulnerability and bypassed network-detection mechanisms to exfiltrate data via a Command and Control server. As a countermeasure, we propose a method called Buffer-Based Signature Verification (BBSV). This approach involves embedding 32-byte tags into files prior to their storage in the cloud, thus offering enhanced data protection. The BBSV method can be integrated into software like MOVEit Secure Managed File Transfer, thereby thwarting attempts by ransomware to exfiltrate data. Empirically tested using a BBSV prototype, our approach was able to successfully halt the encryption process for 80 ransomware instances from 70 ransomware families. BBSV not only stops the encryption but also prevents data exfiltration when data are moved or written from the original location by adversaries. We further develop a hypothetical exploit scenario in which an adversary manages to bypass the BBSV, illicitly transmits data to a Command and Control server, and then removes files from the original location. We construct an extended state space, in which each state represents a tuple that integrates user authentication and system components at the filesystem level.

1. Introduction

In today's digital age, the covert extraction of data represents a major concern across various industries. This challenge is further intensified by the strategic use of encryption by malicious actors. The seriousness of this issue is highlighted by the increasing frequency of ransomware attacks, as seen in notable incidents involving companies like Medibank and Optus in Australia [1]. These attacks are not limited to simply encrypting user data and demanding a ransom for its release. The attackers also threaten to release the stolen data on the dark web if their demands are not met, adding an extra layer of threat.

One significant example of this threat is the MOVEit software, which is widely used in various sectors including healthcare, government, and financial services. MOVEit, known for its file-transfer protocols, automation, analytic, and failover capabilities, was used by entities such as Rochester Hospital and Medibank [2]. The software's broad adoption makes it a notable target for cyber attacks.

These incidents, often exploiting zero-day vulnerabilities, highlight the complex and urgent nature of today's cybersecurity landscape. The threat is multifaceted, encompassing not only the encryption of data for ransom but also the dangerous possibility of sensitive information being leaked if ransom demands are not met. For instance, in February 2024, Lurie Children's Hospital was forced to take its IT systems offline after being targeted by the Rhysida ransomware group, which demanded $3.6 million for the stolen data. This attack disrupted medical services and affected various hospital operations, including internet and email services [3].

* Corresponding author.
E-mail addresses: amahboubi@csu.edu.au (A. Mahboubi), seyit.camtepe@data61.csiro.au (S. Camtepe), keyvan.ansari@murdoch.edu.au (K. Ansari), pawlowski.mp@gmail.com (M. Pawłowski), pawel.morawiecki@gmail.com (P. Morawiecki), haboutorab@csu.edu.au (H. Aboutorab), josef.pieprzyk@data61.csiro.au (J. Pieprzyk), dudajar@gmail.com (J. Duda).

https://doi.org/10.1016/j.jisa.2024.103873

Available online 9 September 2024

This evolving threat landscape necessitates a sophisticated and proactive approach to cybersecurity, emphasising the need for advanced detection mechanisms and robust data-protection strategies.

This paper presents an innovative data-protection approach called Buffer-Based Signature Verification (BBSV), transcending the traditional dependency on statistical/probabilistic methods, and shallow- and deep-learning analyses within the scope of data-storage processes. The BBSV methodology capitalises on the capacity of the filesystem to process data in a buffer-to-buffer manner, simultaneously undertaking the extraction and authentication of the embedded data signature prior to granting permission for data overwrite. This operation is contingent on the detection of possible data-signature manipulation.

Our model introduces a security architecture that uses an encoder–decoder scheme to safeguard against data manipulations. The encoder is embedded at the application's userspace level, marking data with a digital signature to validate its authenticity before they are stored. The decoder is set up at the filesystem level (implemented at the proxy gateway), at which it identifies and processes the digital signature during the data's journey from user space to external storage, such as cloud services or servers. This setup is especially relevant for scenarios in which data are disseminated over network-shared drives.

We assume a system in which data are not stored on local hard drives but entrusted to cloud-based storage systems. To facilitate this, we have designed a proxy gateway that acts as an intermediary between the cloud storage and user space. The key component of this design is the integration of the Filesystem in Userspace (FUSE), enhancing the interaction between local and remote storage systems [4–8]. To assess the effectiveness of our BBSV encoder and decoder, we created a BBSV prototype and tested it against ransomware, conducting simulated attacks with 80 distinct iterations from 70 diverse ransomware families. While our demonstration primarily targeted image data, the underlying architecture of our system is adaptable and suitable for various data types.

Note that in this study, we used a healthcare scenario as our case study, emphasising the critical importance of protecting health data against ransomware attacks. While we used clinical images to demonstrate the clarity and effectiveness of our approach, the BBSV method can indeed be extended to other file formats such as PDFs, DOCs, and PPTs.

Contribution

Key contributions of our BBSV mechanism include the following.

- **Integration with Filesystem Operations:** Embeds 32-byte tags directly in data-write operations, facilitating real-time verification and immediate detection of unauthorised encryption attempts.
- **Deterministic Model for Enhanced Security:** Uses cryptographic signing at the endpoint and verification at a proxy gateway, ensuring data protection based on predetermined, reliable signatures.
- **Prevention of Unauthorised Modifications:** Prevents unauthorised modifications and overwrites through deterministic models and cryptographic signing, significantly enhancing data protection against ransomware attacks.
- **Final Line of Defence in Multi-layered Security Framework:** Acts as a robust final defence, preventing unauthorised data encryption even if initial detection systems are bypassed.

The remaining sections of this study are structured as follows. Section 2 reviews related works and provides the technical background required for the development of a filesystem encoder and decoder, Section 3 delves into formalising ransomware threats, the threat models, and their underlying assumptions. Section 4 details our encoder and decoder scheme in the FUSE. Section 5 discusses the buffer-based signature verification model in Coloured Petri Nets. Section 6 presents the setup, evaluation, and results. Section 7 compares BBSV with the ShieldFS solution while Section 8 provides considerations, a summary of our research, and directions for future research.

2. Related work and background

In the history of computer and communication-network evolution, the ready availability of crypto viruses has triggered numerous malware attacks. A comprehensive examination of these has resulted in a rich repository of academic literature [9]. A crypto virus is a malicious digital entity that employs potent cryptographic algorithms to encrypt data on the targeted device. This ciphered information can only be decrypted using a key that is generated during the malware's operational cycle, thereby preventing the victim's access to their data.

The burgeoning adoption of machine learning (ML) algorithms, including but not limited to Bayesian networks, naive Bayes, the C4.5 decision-tree variant, logistic-model trees, random forest trees, k-nearest neighbours, multilayer perceptrons, simple logistic regression, support vector machines, and sequential minimal optimisation, has been observed in the context of ransomware detection on end-user devices. These methods of detection principally focus on Application Programming Interface (API) calls, anomalous behaviours, and whitelists, in which processes are assigned an anomaly score [10–13].

Despite the considerable success of ML-based techniques in detecting ransomware infections, the purveyors of ransomware continually devise sophisticated tactics to evade detection and circumvent countermeasures. One such innovative approach involves file-less ransomware, which is coded in PowerShell and directly executed in memory, thereby bypassing the need to store the ransomware's binary file on the hard drive. These file-less threats evade detection by employing a technique called reflective dynamic-link library (DLL) injection, which allows for DLL injection from memory rather than the hard drive, presenting a more clandestine method compared to traditional DLL injection.

Some researchers have integrated both static- and dynamic-analysis methods to understand the behaviour and identify features of ransomware. The static analysis seeks to discern whether a given sample qualifies as ransomware by extracting structural information from the sample without executing it. Dynamic analysis involves executing the sample and subsequently observing its behaviour to classify it as ransomware. By monitoring and analysing hardware, filesystem, network traffic, and API call behaviours, a range of ML-based ransomware-detection systems for PCs/workstations have been proposed, employing behavioural features.

A common behavioural feature detected from the dynamic dissection of ransomware is API calls. Several studies [10,14,15] have creatively used these API calls as variables for constructing ML classifiers to identify ransomware threats in personal computers and workstations. A diverse range of methods has been employed across different studies, such as Support Vector Machine (SVM) classifiers, Long-Short-Term Memory (LSTM) classifiers, Recurrent Neural Network (RNN) classifiers, and Restricted Boltzmann Machine classifiers, all capitalising on API calls as features. Other research work has favoured the N-grams of API calls for SVM classifiers and a broad array of ML-based classifiers [16].

In addition to formulating various classifiers through API calls, some researchers have focused on isolating the most impactful API-call features. [17] proposed a novel filtering method for the feature-selection process, aspiring to isolate the most fitting API-call n-grams for ransomware identification and subsequently testing the efficiency of various ML classifiers. Alhawi et al. [15] shifted their focus towards identifying the most significant API-call features and the optimum classifier combination in an ensemble of classifiers, thus facilitating ransomware detection.

Kharaz et al. [18] proposed a dynamic-analysis system called UNVEIL to detect ransomware while it is tampering with a user's files. It generates an artificial user environment and detects ransomware by monitoring I/O requests and filesystem modifications. Continella et al. [19] proposed an add-on driver called ShieldFS that makes the Windows native filesystem resistant to ransomware attacks. They focused on I/O request packets (IRPs) and proposed a custom classifier trained on the filesystem activity based on a combined analysis of write entropy, frequency of read, write, and folder-listing operations, dispersion of per-file writes, a fraction of files renamed, and file-type usage statistics.

Kharaz et al. [20] proposed a minimal modification of the operating system called Redemption to maintain a transparent buffer for all storage I/O. The system monitors the I/O request patterns of applications for signs of ransomware-like behaviour. [21] proposed SSD-Insider++, a new ransomware defence system that protects users' files from cyber attacks using ransomware. SSD-Insider++ is a piece of the firmware found in SSD controllers. SSD-Insider++ combines two innovative features that work in tandem: ransomware detection and perfect data recovery.

Given the prevalence of ransomware attacks and the tendency of cyber criminals to repurpose cybersecurity tools to evade detection mechanisms, a pre-data signature is crucial for validating data and immunising them from illegal encryption before storing them in HD. While current statistical encryption analysis, such as checking file entropy, is limited to protecting data with a high entropy value, signing off data could be a potential solution as ransomware encrypts the data.

The article *REDFISH: Ransomware Early Detection from File Sharing Traffic* [22] presents a novel approach to detecting ransomware activities targeting shared network volumes using a network-traffic inspection device. The detection algorithm, REDFISH, is designed to monitor Server Message Block (SMB) protocol traffic to identify ransomware behaviours such as reading, writing, and deleting files. Unlike traditional anti-malware solutions that require installation on individual hosts, REDFISH operates externally by analysing replicated traffic from a switch port mirror, thereby avoiding additional delays and preventing malware from disabling the detection system. The dataset used for evaluation included more than 50 samples from 19 different ransomware families. Experiments demonstrated a detection success rate of 100%, with alarm triggers occurring within 20 s in most cases, and limiting the ransomware to encrypting a maximum of 10 files before detection and blockage. This method ensures minimal disruption to user activities and enhances the robustness of ransomware-defence mechanisms in corporate environments.

Rcryptect [23] is a real-time detection system for identifying cryptographic operations in a userspace filesystem. The approach leverages FUSE and the NIST randomness-test suite to monitor and detect high-entropy blocks indicative of cryptographic activity. The dataset includes text files, MP3s, JPEGs, and ZIPs, gathered from various online sources, with 1000 files for training and 200 for testing each type. The method's core achievement is its ability to distinguish encrypted from normal blocks using statistical analysis, thereby preventing ransomware attacks with minimal false positives. Rcryptect adapts file I/O processes by customising write and delete operations to ensure that only authorised users can perform these actions on suspicious blocks. The methods encompass the use of statistical frequency tests, entropy measurements, and heuristic rules to identify cryptographic operations, all implemented without kernel modifications, making the system versatile across different operating systems.

Rcryptect required an overhead of approximately 10%–15% in writing time compared to a naive FUSE, with specific delays for different file types being 12.3% for text, 15.1% for MP3, 15.3% for JPEG, and 12.2% for ZIP. This overhead was consistent across various platforms, including MacOS and Windows 10.

The article *Ransomware Detection based on Machine Learning using Memory Features* by Malak Aljabri et al. [24] presents an approach to ransomware detection leveraging ML and memory forensics. The authors built a new dataset comprising samples from recent ransomware groups (Revil, Lockbit, BlackCat) and benign samples (Office applications, Windows applications, compression applications). These samples were dynamically analysed in an enhanced Cuckoo sandbox to ensure reliability. They experimented with several ML models, including Random Forest, LightGBM, Adaptive Boosting, Extra Tree, and XGBoost, and found XGBoost to be the most effective, achieving 97% accuracy with a 2% false-positive rate using 47 out of 58 features. This research highlights the importance of memory traces in detecting ransomware and provides a robust model for recognising unknown ransomware samples. The method involves detailed steps such as dataset construction, feature extraction, feature selection using Chi-Squared and Sequential Forward Selection, model training, and performance evaluation, emphasising the model's accuracy and low false-positive rate as key achievements.

The paper *Synthesizing Dynamic Ransomware via GAN* investigated the robustness of machine-learning-based ransomware detectors by generating adversarial ransomware samples using Generative Adversarial Networks (GANs) [25]. The dataset comprised dynamic execution logs from various ransomware families, such as Locky and Cerber, collected via VirusTotal, and tagged by Microsoft and Kaspersky. The samples were executed in a Windows environment, and their behaviours were captured using the .Net FileSystemWatcher API. The benign data included logs from the installation and execution of various applications and idle system activities. Feature mapping categorised I/O events and entropy changes, and the data were pre-processed into 28 × 28 2D arrays to facilitate transfer learning from vision-based GANs. An Auxiliary Classifier GAN (ACGAN) was trained to produce adversarial samples, which were then assessed for quality using a proposed adversarial quality metric based on n-gram statistical similarity.

The experimental results revealed significant vulnerabilities in current ransomware detectors. While initial testing showed that Text-CNN achieved the highest accuracy (99%) and the lowest false-positive rate (3%), the evaluation against adversarial samples demonstrated a drastic performance decline. Most classifiers, including Text-CNN, LDA, Naive Bayes, and SVM-linear, failed to detect any adversarial samples, with only Random Forest and SVM-radial showing some resilience (36% and 100% detection rates, respectively). Analysis of the Text-CNN latent-feature subspace indicated that adversarial samples were misclassified as benign, exposing blind spots in the model. This study underscores the need for robust ML models that incorporate adversarial training and comprehensive evaluations to enhance security against sophisticated attacks.

Integrated Detection and Mitigation of Ransomware Through the Filesystem [26] introduces GuardFS, a framework designed to detect and mitigate ransomware attacks on Linux-based systems. This approach leverages an overlay filesystem to monitor system calls, which are then analysed using ML techniques to classify processes as benign or malicious. The methods employed include deploying the framework on a Raspberry Pi acting as an FTP server and using three ransomware samples (RansomwarePoC, DarkRadiation, and roar) to evaluate performance. Data from system calls, including read and write operations, were collected and processed to extract features such as Shannon entropy, measuring the randomness of the data. The dataset was split into training and evaluation sets, with models trained using algorithms such as Random Forest Classifier, Logistic Regression, and Isolation Forest. The experiments, conducted in a virtual test bed with eight different ransomware samples and seven defence scenarios, demonstrated the framework's effectiveness in significantly reducing data loss.

The results indicated that GuardFS effectively mitigates ransomware attacks, with the best defence configuration (DEL+OBF with a 5-s delay) resulting in an average data loss of only 703 KB compared to 7.43 GB with no defence. The analysis showed that the DEL+OBF strategy, while introducing some overhead, provided the strongest defence, significantly reducing data loss by delaying and obfuscating ransomware operations. The PKILL defence, which terminates malicious processes, was the most resource-efficient but was less effective in preventing data loss. Overhead analysis revealed that, while the PKILL defence incurred minimal resource consumption, the DEL+OBF strategy required more computational resources due to its delay and monitoring mechanisms. However, the increased resource consumption was justified by the enhanced security and significant reduction in data loss, making GuardFS a robust solution for protecting Linux-based systems against ransomware attacks.

The article *DeepWare [27]: Imaging Performance Counters With Deep Learning to Detect Ransomware* presents a novel approach to ransomware detection by converting hardware performance counter (HPC) data into behavioural images and using a Convolutional Neural Network (CNN) for classification. The method involves monitoring system-wide changes in HPC data distribution, capturing these changes as images, and feeding them into a CNN to distinguish between ransomware and benign activities. The dataset comprised 420,000 behavioural images, evenly split between ransomware and benign activities, ensuring a balanced training and testing process. The study used a 10-fold cross-validation technique to evaluate the model's robustness. DeepWare significantly outperformed existing models such as OC-SVM, RATAFIA, and EGB, achieving a 99% recall rate and nearly zero false-positive rates with just a 100-millisecond snapshot of the HPC data. The CNN architecture was specifically tailored to handle the unique properties of the HPC data, incorporating zero-padding and fast-convolutional operations to extract relevant features from the behavioural images.

The filesystem plays a crucial role in the DeepWare study, serving as a verification tool to ensure accurate data collection during ransomware attacks. The researchers used a filesystem watcher to monitor modifications to documents, providing visual clues such as changes in RAM and CPU usage, and the appearance of dropped ransom notes. This allowed the team to pinpoint the exact time ransomware started encrypting files, ensuring the accuracy of the labelled data. Each monitored hardware event's aggregate count was sampled every 10 ms and saved to a file, creating a detailed log of system activity during both ransomware and benign operations. This meticulous data-collection process was essential for training the DeepWare model, enabling it to learn and identify the distinct behavioural patterns associated with ransomware attacks effectively. By leveraging filesystem monitoring, the study ensured that the dataset accurately reflected real-world ransomware behaviour, enhancing the model's ability to detect both known and zero-day ransomware threats.

L-IDS: A Multi-Layered Approach to Ransomware Detection in IoT [28] introduces L-IDS, a lightweight intrusion-detection system designed to address ransomware threats in resource-constrained IoT environments. The approach combines multiple protective and detective techniques, including ML-based anomaly detection, decoy files, entropy measurement, fuzzy hash measurement, and Trusted Execution Environment (TEE). The dataset used for evaluation consisted of over 667,000 malware and benign files, with a focus on 1555 ransomware samples from various families such as Azov, Conti, and Dharma. The method incorporated Gaussian Naive Bayes (GNB) for anomaly detection, supported by decoy files and entropy measurements, to detect abnormal file and network behaviour indicative of ransomware. The results demonstrated that L-IDS achieved a high detection accuracy, outperforming traditional signature-based (ClamAV) and behaviour-based (OSSEC) detection methods, with L-IDS identifying 100% of the ransomware samples in real-time while maintaining low resource consumption. This multi-layered defence strategy enhances the security of IoT systems by effectively detecting and mitigating both known and unknown ransomware attacks.

The study *Ransomware over Browser (RØB): An Attack Vector* [29] explores a newly identified ransomware threat vector that leverages the filesystem Access (FSA) API to execute browser-based ransomware attacks. The researchers implemented three distinct methods to analyse and mitigate this threat. The first method involved creating JavaScript hooks for the FSA API to monitor and prevent malicious file modifications before they become permanent. This approach used ML classifiers trained on features such as entropy change and file-size change, demonstrating high accuracy in detecting ransomware activities. The second method focused on local activity monitoring, in which the researchers collected and analysed data from FSA API function calls, system calls, and filesystem activities. They employed n-gram analysis to differentiate between benign and malicious behaviours, effectively identifying patterns indicative of ransomware attacks. The third method proposed a redesign of the FSA API permission dialogues to enhance user awareness and decision-making, incorporating explicit warnings and detailed information about file access and modifications.

The datasets used in this study included both benign and malicious file modifications, generated through various operations and configurations of the RØB ransomware. The benign dataset comprised typical file operations performed by web applications, while the malicious dataset included non-adaptive and adaptive versions of RØB, designed to test the robustness of the detection mechanisms. The results highlighted the effectiveness of the proposed defence mechanisms. The API hooking and classification approach achieved high detection accuracy with ML classifiers such as Random Forest and K-nearest Neighbour. Local activity monitoring successfully identified ransomware activities through pattern recognition, although some adaptive attack strategies posed challenges (see Table 1).

3. Ransomware threat formulation and the threat model

The escalation in the sophistication of ransomware attacks has necessitated a paradigm shift in how defence mechanisms are constructed and implemented. This is because ransomware developers employ a wide variety of techniques, such as memory-mapped I/O, together with intermittent and partial encryption techniques, to minimise the chances of successful attack detection using probabilistic and traditional defence mechanisms. Such approaches in the phase of data encryption can lead to successful attacks. Therefore, in setting up the threat model here, we first formalise the ransomware tactics and defence methods.

3.1. Ransomware variants and families

The classification of ransomware operations into distinct categories provides a structured approach to analysing the cybersecurity threat landscape. By denoting the set of known ransomware variants as $R = \{r_1, r_2, \dots, r_n\}$ and known ransomware families as $F = \{f_1, f_2, \dots, f_m\}$, we establish a catalogue for existing threats. In recognising the dynamic nature of malware, we extend this cataloguing to unknown entities, with $U = \{u_1, u_2, \dots, u_k\}$ representing unknown ransomware families and $V = \{v_1, v_2, \dots, v_l\}$ unknown ransomware variants.

This process is further refined by distinguishing the known and unknown tactics used by ransomware developers, represented by $T = \{t_1, t_2, \dots, t_p\}$ and $W = \{w_1, w_2, \dots, w_q\}$, respectively. This distinction is crucial to the strategic development of cybersecurity defences, acknowledging both the current threat environment and the potential for unforeseen adversarial tactics.

Probabilistic modelling serves as a critical tool in this framework, with $P(U)$, $P(V)$, and $P(W)$ denoting the respective probability distributions over sets of unknown ransomware families, variants, and tactics, respectively. This approach enables a quantitative assessment of risks and informs the allocation of defencive resources to mitigate potential threats effectively.

Moreover, operational methods are systematised by defining a function $\text{opMethod} : (R \cup V) \to M$, associating each ransomware variant with a specific operational method, known or unknown. This association aids in the development of defencive measures and the subsequent forensic investigation of ransomware incidents.

3.2. Dynamic-threat model

Threat models and landscape analysis lies in the systematic and quantitative assessment of cyber threats. A dynamic threat model is essentially a mathematical representation that encapsulates the complexities of cyber threats by correlating various elements such as tactics T, techniques W, ransomware R, variants V, and their respective

Table 1
Comparison of the proposed BBSV method with existing methods. BBSV uses a deterministic method, avoiding the use of probabilistic models for the detection of ransomware behaviour and patterns within the system. BBSV is the last line of defence against unknown ransomware.

Feature	UNVEIL	ShieldFS	Redemption	SSD-Insider++	GuardFS	BBSV
Dynamic Analysis	✓	✓	✓	✗	✓	✗
Static Analysis	✗	✗	✗	✗	✗	✗
File-less Ransomware Detection	✗	✗	✗	✗	✗	✓
API Call Monitoring	✓	✓	✓	✗	✓	✗
High-Entropy Detection	✓	✗	✗	✗	✗	✗
Pre-data Signature Validation	✗	✗	✗	✗	✗	✓
I/O Request Monitoring	✓	✓	✓	✓	✓	✗
Perfect Data Recovery	✗	✗	✗	✓	✗	✗

probabilities P to yield potential countermeasures C. This formulaic approach allows for the inclusion of both known (T, R) and speculative or emerging (W, V) components, thereby ensuring a comprehensive framework that can adapt to evolving threats. It is necessary to analyse and quantify the threat posed by both known and unknown ransomware families, variants, and tactics.

- This analysis can be represented by the function

  $$\text{Analysis} : (F \cup U) \times (R \cup V) \times (T \cup W) \to I,$$

  where F represents known threat factors, and U unknown or emerging threat factors.
- The indicator set I can be broken down into:
  - I_{threat} Threat indicators, which could include metrics such as frequency of attacks, severity of attacks, etc.
  - I_{vul} Vulnerability indicators, which could include metrics such as the number of unpatched systems, the number of systems running outdated or unsupported software, etc.
  - I_{res} Resilience indicators, which could include metrics such as the time to detect an attack, the time to respond to an attack, the effectiveness of the response, etc.
- These indicators $I = \{I_{\text{threat}}, I_{\text{vul}}, I_{\text{res}}\}$ can then be used to inform and update the Dynamic-Threat Model and to develop, evaluate, or refine the countermeasures C aimed at mitigating the threat of ransomware. In this paper, we focus on the resilience-indicator subset I_{res}.

3.3. Formal description of the ransomware-counteraction strategy

Operational methods related to tactics employed by ransomware developers can be further defined using the function *opMethod_Relation*, which characterises the relationship between operational methods (M) and the combined tactics, denoted as weaponry $(T \cup Wp)$. This function maps these elements onto a measure of effectiveness, aiding in the evaluation of operational methods against various ransomware tactics.

Definition of the Weaponry Set

Let $Wp = \{w_1, w_2, \dots, w_{p_n}\}$ denote the set of known or unknown tactics employed by ransomware developers.

Definition of the Function

$$\text{opMethod\_Relation} : M \times (T \cup W\,p) \to E,$$

where M is Operational Methods, T Tactics, Wp Weaponry (as defined above) and E an Effectiveness Measure.

Variable Difficulty due to Wp Changing

Weaponry Wp is subject to change over time, so we denote it as $Wp(t')$ to explicitly indicate its time dependence. This results in the function *opMethod_Relation* becoming time-dependent as well:

$$\text{opMethod\_Relation} : \left(M, T \cup Wp(t')\right) \to E(t').$$

Expression of the Challenge

The challenge in evaluating the effectiveness arises due to the dynamic nature of $Wp(t')$, making it difficult to ascertain the relation or measure of effectiveness consistently. This variability introduces uncertainty in the estimation of $E(t')$; this can be expressed as

$$\Delta E(t') = E(t'_1) - E(t'_0).$$

$\Delta E(t')$ represents the change in effectiveness due to the modifications in weaponry over time.

To counter the change in effectiveness $\Delta E(t')$, we propose the incorporation of BBSV encoders and decoders in the defence model. This strategic enhancement of I_{res} specifically targets the $Wp(t')$, for example, the new obfuscation and intermittent/partial encryption subtleties common in contemporary ransomware, thereby bolstering the defence mechanism's anticipatory and mitigative capacities against evolving ransomware threats.

3.4. System overview

The BBSV method embeds cryptographic signatures in data to facilitate the detection of subtle data alterations indicative of a ransomware attack. This innovation in detection technology shows promise in offering a more reliable differentiation between legitimate and malicious encryption processes, a crucial aspect for timely threat identification.

Furthermore, harnessing the capabilities of the FUSE to create a resilient filesystem module acting as a pass-through proxy is an additional layer of defence aimed at enhancing the model's resilience against adaptive threats I_{res}. This inclusion amplifies the defence model's efficacy in tackling ransomware attacks, especially those employing innovative encryption techniques such as partial and intermittent encryption, as observed in the BlackBasta ransomware. The integration of FUSE is indicative of a holistic approach, one that seeks to fortify the defence framework against attacks from multiple angles.

The following are areas in which BBSV offers potential benefits to I_{res}.

- *Enhanced Security for Sensitive Operations*: BBSV is designed to add an extra security layer, acting as a gateway that scrutinises data at the block level before they are written from the endpoint to the cloud storage, potentially catching ransomware-related alterations. This is true when organisations rely on SMB or NFS to share the data with endpoints.
- *Decoupling Authentication from Cloud Providers* [30]: By using a proxy for authentication, BBSV can act as an independent verification point, separate from the cloud provider's systems. This approach could be beneficial if the cloud provider's authentication systems are compromised. However, the effectiveness of this strategy depends on the organisational Information and Communications Technology architectures, and whether BBSV is integrated as part of cloud services or implemented as a standalone system.
- *Reducing Trust in Endpoints*: BBSV can limit the degree to which a compromised client can affect cloud storage. Even if a client's primary authentication mechanism is compromised, BBSV can provide a secondary check that prevents unauthorised data modification.

- *Protection Against Stolen Credentials*: If an attacker obtains a user's cloud storage credentials or run-time attacks in memory (e.g. Mimikatz), they could potentially bypass the endpoint or cloud provider's MFA if they have access to the required authentication factors. BBSV can introduce a separate defence barrier for adversaries to overcome, such as verifying file transfers between endpoints and storage by ensuring that the file has been signed and by detecting signatures before proceeding with write or overwrite actions at the storage.

4. Designing buffer-based signature verification

Our proposed model, BBSV, is designed to facilitate robust data-protection measures against illegal data encryption, especially ransomware, specifically for image data and compressed data formats. We premise this model on the assumption of a discrete data source composed of a finite set of Unicode characters, denoted $S = \{s_1, s_2 \cdots s_i\}$. The data are stored in block-oriented storage media, which is representative of diverse storage technologies such as NAND flash memory, solid-state drives, and hard disk drives. In block-oriented storage systems, data transfers are executed in fixed byte quantities, even if the requirement is for a single byte. Read/write operations usually use either 512 bytes or 4096 bytes, depending on the system configuration and the operating system. Here, we consider a scenario in which data transfer is performed in 4096-byte blocks. Under these assumptions, the data set S can be constituted as b_n read/write buffers, with each buffer $b_0 \cdots b_{n-1}$ having a fixed length of 4096 bytes. The final buffer b_n has a length constraint: $0 < b_n \leq 4096$ bytes.

To augment the resilience of this storage system against ransomware attacks, we introduce a cryptographic layer using both the ChaCha20 cipher and the Poly1305 authentication algorithm, as specified in RFC7539. This layer incorporates the standalone use of the ChaCha20 stream cipher and the Poly1305 authenticator, and also their combined use within an Authenticated Encryption with Associated Data (AEAD) algorithm. ChaCha20 is a software-oriented high-speed stream cipher, which operates approximately three times faster than AES on platforms lacking specialised AES hardware [31]. Poly1305 is a one-time authenticator, producing a 16-byte tag from a given 32-byte one-time key and a message. This tag is instrumental in validating the integrity of data buffers.

The core components of BBSV include encoding and decoding submodules. The encoding submodule processes digital data (e.g. an image), generates digital signatures, encrypts the image, and subsequently stores it in the cloud storage. The decoding submodule decrypts and validates digital signatures prior to any modification or encryption action by an application on the image data.

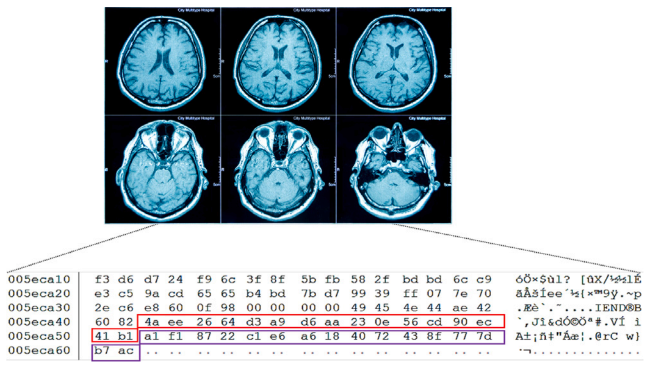
4.1. BBSV encoder

We employ the ChaCha20 cipher to ensure data confidentiality before data are transmitted to the cloud. In this process, a 256-bit (32-byte) key k and a 128-bit (16-byte) nonce v are used. S denotes as a 512 S-byte sequence, with a length of l data units. The ChaCha20 encryption of the sequence $s_0, \dots, s_i$ with nonce v under key k, is represented as $\text{ChaCha20}_k(k, v) \oplus S_i$, and yields an l-byte sequence.

In the first step of our encoding algorithm, we divide the unencrypted buffers into two temporary lists, $temp_1$ and $temp_2$, based on their lengths (Algorithm 1, lines 9–15). The first list, $temp_1$, holds buffers of size 4096 bytes, while $temp_2$ holds any remaining buffer that is less than 4096 bytes in size.

Next, we use ChaCha20 to encrypt the buffers in *temp_1* (Algorithm 1, line 17) and produce the encoded sequence $c_0 \cdots c_{n-1}$. Following this, Poly1305 is used to generate a tag Tg_1 from $c_0 \cdots c_{n-1}$ (line 19), which is then concatenated with $temp_2$ (line 21).

After this, ChaCha20 is applied again to encrypt the updated $temp_2$ (line 21), which now includes the first payload Tg_1. Subsequently, a



```
005eca10   f3 d6 d7 24  f9 6c 3f 8f  5b fb 58 2f  bd bd 6c c9   óÖ×$ùl? [ûX/½½lÉ
005eca20   e3 c5 9a cd  65 65 b4 bd  7b d7 99 39  ff 07 7e 70   ãÅšÍee´½{×™9ÿ.~p
005eca30   2e c6 e8 60  0f 98 00 00  00 00 49 45  4e 44 ae 42   .Æè`.˜....IEND®B
005eca40   60 82 4a ee  26 64 d3 a9  d6 aa 23 0e  56 cd 90 ec   `‚Jî&dÓ©Öª#.VÍ ì
005eca50   41 b1 a1 f1  87 22 c1 e6  a6 18 40 72  43 8f 77 7d   A±¡ñ‡"Áæ¦.@rC w}
005eca60   b7 ac .. ..  .. .. .. ..  .. .. .. ..  .. .. .. ..   ·¬..............
```

Fig. 1. An image incorporating two 16-byte modulated payload tags at the end. It should be noted that, from a visual standpoint, this image was not encrypted with ChaCha20.

second tag Tg_2 is derived using Poly1305 (line 24) and concatenated to $temp_2$ (line 25). Finally, the encrypted buffers are written to the storage as an output of the encryption process (line 26).

In this method, two 16-byte payloads (Tg_1 and Tg_2) are embedded in the original data. In the context of an image, this would be akin to adding 4 pixels wide × 2 pixels high to the original size of a 300 DPI image that is 412 pixels wide × 324 pixels high. Figs. 1 and 2 illustrate the process of adding tag payloads to an image. Algorithm 1 describes the entire process in detail.

Algorithm 1 Encoding submodule algorithm that embeds two 16-byte signatures (tag 1 and 2) as file payloads.

```
1: Input: $buf$ : [$b_0^0$ , $b_0^1, \cdots$ , $b_n^i$] % Unencrypted Buffers
2: Outputs: Encrypt and adding watermark tags as payloads.
3: ChaCha20_Secret_Key # A 256-bit key
4: nonce # A 128-bit nonce
5: key # Poly1305 32 bytes key
6: $temp_1$ =: []
7: $temp_2$ =: []
8: for each $b_n^i$ in $buf$ do
9:     $ln =: length(b_n^i)$
10:    if $ln = 4096$ then
11:        $temp_1.append(b_n^i)$
12:    else if $ln < 4096$ then
13:        $temp_2.append(b_n^i)$
14:    end if
15: end for
16: # Encrypting buffer
17: $temp_1$ := ChaCha20($temp_1$, key, nonce)
18: # Generate signature Tag 1
19: $b_n^{Tg1}$ := Poly1305.generate_tag(key, $temp_1$)
20: # Concatenating $b_n^{Tg1}$ to $temp_2$ and encrypt
21: $temp_2.append(b_n^{Tg1})$
22: $temp_2$ := ChaCha20($temp_2$ , key, nonce)
23: # Generate signature Tag 2
24: $b_n^{Tg2}$ := Poly1305.generate_tag(key, $temp_2$)
25: $temp_2.append(b_n^{Tg2})$
26: write($temp_1$ | $temp_2$)
```

4.2. BBSV decoder

Consider a filesystem in which the write function accepts an input sequence of n buffers, with each buffer having a block size b_l of 4096 bytes (equivalently 2^{12} bytes). Suppose an application seeks to read, decrypt, view, re-encrypt, and then overwrite an image file that contains the two digital signatures embedded by the encoder.

Before the decoded image can be written back to the storage in the cloud, the system must generate and validate signature tags matching those embedded in the image file. The decoder accomplishes this by reading encrypted buffers from memory and storing them in two temporary lists. Let C contain buffers signed with a fixed number N of inputs. A 32-byte segment of C consists of Poly1305 tags, denoted

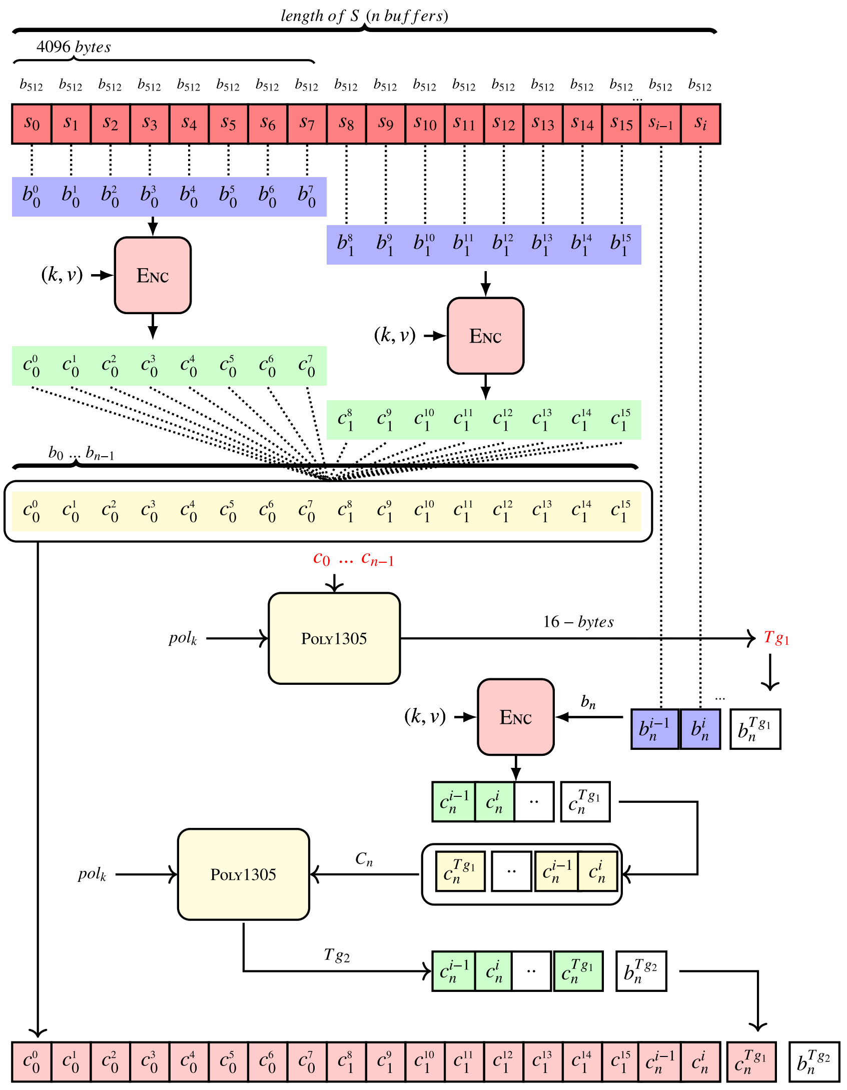


Fig. 2. BBSV encoder submodule ChaCha20-Poly1305 encryption module.

as Tg_1 and Tg_2. Tg_1 is encrypted by the encoder while Tg_2 is not. We represent the total length of these two tags as l_{tg}, such that $l_{tg} = l_{tg}(Tg_1, Tg_2)^{l_{tg}}$ are 32 bytes. Furthermore, we denote $c_0, c_1, \dots, c_n{}^l$, as incoming buffers where $l = l(n)$, the total byte length of C. To determine the total number of blocks b_l in C (each b_l is 4096 bytes long), we compute $c_{n_{b_l}} = \lfloor (l - l_{tg})/b_l \rfloor$.

The decoder sequentially reads incoming buffers in blocks of b_l into a temporary list: $temp_1 \leftarrow [c_0^0, \dots, c_0^7, \dots, \dots, c_n^{i\,b_l}]$. The buffer c_n^i is recognised as the last buffer of C. If the size of $c_{n-1}^i, \dots, c_n^{i\,b_l}$ is less than b_l, we should first check if $b_n \geq 32$. If true, then these elements are stored in the second temporary list: $temp_2 \leftarrow [c_{n-1}^i, \dots, c_{n_{(b_n)}}^i]$, where b_n is the size of the last block buffer in C. This dual-list structure allows for efficient handling of both full-size and residual buffers during the read and write operations.

Upon receiving $temp_1$ and $temp_2$, the decoder proceeds to identify the two signatures. We first examine $temp_2$, which represents the last block of C, and assign the byte length of $temp_2$ to n. The condition $b_n \geq 32$ must hold.

The decoder input for $temp_2$ is $C_{temp_2} = c_0 \| c_1 \| \cdots \| b_0 \| b_1 \| \cdots$, where c stands for ciphered bytes and b for unencrypted bytes. The decoder, having prior knowledge that the last 16 unencrypted bytes of $C_{temp_2}[n]$ contain a tag that validates $C_{temp_2}[n-16]$ bytes, extracts $Tg_{2'}$ by slicing the list as $temp_2[n-16 : n]$. The integrity of $temp_2[0 : n-16]$ bytes is verified using a Poly1305 function, $f_h(input, tag, key)$. In our case, the inputs are $f_h(temp_2[0 : n-16], Tg_2, k')$. This function returns 1 if the integrity check passes or 0 otherwise:

$$f_h = \begin{cases} 1, & \text{if } Tg_2' = Tg_2'' \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Assuming that $f_h(temp_2[0 : n-16], tag_2, k')$ returns true, the decoder decrypts $temp_2[0 : n-16]$ to obtain Tg_1'. Here, k' represents a 256-bit key and v' a 128-bit nonce, both securely exchanged between the encoder and the decoder. The decoder extracts $b_n^{Tg_1}$ from $temp_1$ after validating Tg_2' with $temp_2$. Following this, Tg_1' is used to validate against illegal encryption in $temp_1[n]$ bytes using the Poly1305 function $f_h(temp_1[0], tag_1, k')$. This function raises an exception if the calculated tag does not match with $b_n^{Tg_1}$, thereby indicating a possible unwanted file encryption.[1]

In a communication-channel setup, the encoder and decoder could be positioned at either end, with the responsibility for checking for illegal encryption at the source and destination filesystems, respectively.

[1] Unwanted encryption means that the user did not authorise and sign legitimately through the encoder process.

Algorithm 2 Buffer-based Signature Verification Model decoder submodule algorithm that extracts two 16-byte signatures (tag 1 and 2) from file payloads and validates signatures for any possible unwanted file encryption.

```
Input: $buf$ : [$c_0^0$ , $c_0^1, \cdots ,$ $c_n^i$, $b_n^{Tg_1}$ , $b_n^{Tg_2}$] {# Encrypted buffer}
Outputs: Validate tag payloads.
ChaCha20_Secret_Key {# A 256-bit key}
nonce {# A 128-bit nonce}
key {# Poly1305 32 bytes key}
n =: 0 {$\#offset$}
$temp_1$ =: []
$temp_2$ =: []
$b_n^{Tg_1}$ , $b_n^{Tg_2}$ =: [] {$\#Initializinglists$}
while $True$ do
    Read $c_n^i$ from the $encrypted\ buffer$
    $ln =: length[c_n^i]$
    if $ln = 4096$ then
        $temp_1$ =: [$c_n^i$]
    else if $ln < 4096$ and $ln \geq 32$ then
        $temp_2$ =: [$c_n^i$]
        $break$
    end if
end while
$a =: length(temp_2)$
$b = a - 16$ {   # read 16 bytes from end of the buffer}
for tag2 in range (b , a ) do
    $b_n^{Tg_2} \leftarrow b_n^{Tg_2} \cup \{temp_2[tag2]\}$ {Append $temp_2[tag2]$ to $b_n^{Tg_2}$}
end for
$temp_2 = temp_2[0 : b]$ {# remove $b_n^{Tg_2}$ from $c_n^i$ }
{# Verifying Poly1305 tag 2}
Compute $tag = f_h(temp_2[0 : n-16], Tg_2', key)$ {Poly1305 function}
if $tag == 1$ then
    {# Extract $b_n^{Tg_1}$}
    $i =: length(temp_1)$
    $j = i - 16$ {# read 16 bytes from end of buffer}
    for tag1 in range (j , i ) do
        $b_n^{Tg_1} \leftarrow b_n^{Tg_1} \cup \{temp_1[tag1]\}$ {#Append $temp_1[tag1]$ to $b_n^{Tg_1}$}
    end for
    $temp_1 = temp_1[0 : j]$ {# remove $b_n^{Tg_1}$ from temp_1}
    $Tg_1\_decrypted = ChaCha20\_Decrypt(b_n^{Tg_1}, nonce, ChaCha20\_Secret\_Key)$
    Compute $tag = f_h(temp_1[0 : n-16], Tg_1', key)$ {#Poly1305 function}
    if $tag == 0$ then
        Raise Exception("Invalid Signature. Value did not match computed tag.")"
    end if
else
    Raise Exception("Invalid Signature. Value did not match computed tag.")
end if
```

4.3. BBSV key generation

ChaCha20, a potent cipher, offers a sturdy 256-bit encryption. Concurrently, Poly1305, an authentication algorithm, is constructed to refute inauthentic messages with a probability of $1-\frac{n}{2^{102}}$ for a message comprising of $16n$ bytes, even after the transmission of 2^{64} legitimate messages. Therefore, it is coined as SUF-CMA (Strong Unforgeability against Chosen-Message Attacks) in the cryptographic lexicon [32]. The rigorous proofs for these intricate cryptographic mechanisms, however, exceed the scope of this paper. Interested readers can refer to the associated academic literature [33–35].

A paramount aspect of securely implementing the encoding module is the uniqueness of the *nonce* used in both ChaCha20 and Poly1305. It is anticipated that an attacker employing ransomware might exploit open-source post-exploitation tools (e.g. Mimikatz) to covertly extract stored authentication credentials such as memory-resident passwords, hashes, PINs, and Kerberos tickets. Consequently, the attacker could obtain the session key and nonce, adaptively sign, and then integrate the tags into the encrypted payload during the attack, thereby circumventing the encoder's native functions. In essence, the malefactor could misuse the signer nonce and key to continuously sign data throughout the attack. Therefore, the Trusted Platform Module (TPM) is leveraged to generate a nonce and a session key in a secure manner.

The TPM, embedded with an RSA cryptographic engine and Platform Configuration Registers (PCRs) for preserving cryptographic hashes, otherwise known as integrity measures, plays a pivotal role. Each TPM is equipped with an Endorsement Key, a public–private encryption-key pair supplemented with a certificate from the TPM manufacturer, instrumental for TPM administrative tasks like 'taking ownership' of the TPM. During initialisation, the TPM creates a confidential element *tpmProof*, used to protect the key pairs it generates. TPM 2.0, now an industry-standard in Windows 11 and Windows Server 2022, incorporates NIST-approved cryptographic algorithms, a hierarchy of key management, root keys, authorisation mechanisms, and Non-Volatile Random Access Memory (NVRAM).

In our proposed model, a user-specific request (Req_K) initiates Algorithm 3, which in turn forwards it to the TPM. If the user is not one of the TPM's known users, the algorithm generates a cryptographic challenge for the user and sends it to them. If the user is recognised, a cryptographic session key and nonce are generated by the TPM and sent to both the user and the proxy server. The algorithm then awaits a response from the user to verify if they successfully resolved the challenge. If the challenge response is authenticated, approved user credentials are accepted, and the user's session begins. If the challenge is not authenticated, the request is denied. The algorithm reads a buffer from the filesystem for processing, which entails iterating through each segment of the buffer, represented as b_n^i, where n is the segment index and i indicates the ith segment.

Segments that are exactly 4096 bytes in length are processed with the ChaCha20 cryptographic function using the predetermined session key and nonce. A tag is then produced using the Poly1305 algorithm. Segments that are less than 4096 bytes in length are concatenated with the previously generated tag, and encrypted again with ChaCha20; a new tag is produced, concatenated with the segment, and then encrypted again with ChaCha20. These processed segments are then sent to the proxy server, which extracts the tag from the received segment and verifies it against the segment without the tag using Poly1305. If the verification is successful, the segment is decrypted using ChaCha20 and written to file storage. If the verification fails, an alert is sent to the system administrator.

Let U denote the user initiating the request, *TPM* the Trusted Platform Module (an external node), R_K the user's request, S the set of known users in TPM, C the challenge sent to the user, Res the user's response, K the session key, N a nonce, and D the unencrypted data accessed by the user. Further, let T_1 and T_2 denote temporary buffer storage locations, b_n^i each element in D, and l the length of the b_n^i. Additionally, *Enc* is the encoded data, *Dec* the decrypted data, *ChaCha20* a cipher function, and *Poly1305* a cryptographic message-authentication function. The symbols $\oplus$ and $\|$ respectively denote the concatenation operation and the order of buffer arrangement.

Algorithm 3 Revised TPM, Challenge Authentication, Key Distribution, and Encoding

```
$U \leftarrow initiateRequest()$
$R_K \leftarrow createRequest(U)$
$sendRequestToTPM(R_K)$
if $U \notin S$ then
    $C \leftarrow createChallenge(U)$
    $sendChallengeToUser(C, U)$
else
    $(K, N) \leftarrow TPM.generateSessionKeyAndNonce(U)$
    $sendKeyAndNonceToUser(K, N, U)$
    $updateProxyServer(K, N, U)$
end if
$Res \leftarrow userResponse(U)$
if $verifyChallengeResponse(C, Res)$ then
    $(K, N) \leftarrow TPM.generateSessionKeyAndNonce(U)$
    $sendKeyAndNonceToUser(K, N, U)$
    $updateProxyServer(K, N, U)$
else
    $denyRequest(U)$
    return
end if
$D \leftarrow accessRequestedData(U, K, N)$
$T_1, T_2 \leftarrow [], []$
for each $b_n^i$ in $D$ do
    $l \leftarrow length(b_n^i)$
    if $l = 4096$ then
```

```
            $T_1 \leftarrow [b_n^i - 1]$
        **else if** $l < 4096$ **then**
            $T_2 \leftarrow [b_n^i]$
            **break**
        **end if**
    **end for**
    $N_1 \leftarrow$ *generateNewNonce()*
    $T_1 \leftarrow ChaCha20(T_1, K, N_1)$
    $b_n^{Tg_1} \leftarrow Poly1305.generate\_tag(K, T_1)$
    $T_2 \leftarrow [T_2 \oplus b_n^{Tg_1}]$
    $N_2 \leftarrow$ *generateNewNonce()*
    $T_2 \leftarrow ChaCha20(T_2, K, N_2)$
    $b_n^{Tg_2} \leftarrow Poly1305.generate\_tag(K, T_2)$
    $T_2 \leftarrow [T_2 \oplus b_n^{Tg_2}]$
    $Enc \leftarrow T_1 \parallel T_2$
    *sendEncodedDataToProxyServer*$(Enc, U)$
    **if** verifyTag$(Enc, U)$ **then**
        $Dec \leftarrow$ *decryptData*$(Enc, K)$
        *storeData*$(Dec)$
    **else**
        *notifyAdmin*$(Enc, U)$
        *denyRequest*$(U)$
    **end if**
    *expireKey*$(K, U)$
```

4.4. Overview of challenge-response authentication process

Algorithm 3 employs a robust challenge-response mechanism to authenticate users and ensure secure data transactions. When a user initiates a request, it is first sent to the TPM for validation. If the user is not recognised, the TPM generates a unique cryptographic challenge, which is sent to the user. The user must respond correctly to this challenge using their credentials, which the TPM verifies before granting access. This step ensures that only authenticated users can proceed, adding an essential layer of security.

The TPM then generates a session key and nonce, which are crucial for encrypting the communication between the user, TPM, and the proxy server. These cryptographic elements safeguard against unauthorised access by ensuring that only sessions initiated by verified users can proceed. Data is divided into segments and encrypted using the ChaCha20 cipher, while Poly1305 tags are generated to verify data integrity. The proxy server checks these tags before decrypting and storing the data, alerting administrators if any discrepancies are detected.

This process is further illustrated in Fig. 3, and demovideo which provides a visual representation of the challenge-response workflow. The inclusion of these detailed steps clarifies the algorithm's function and highlights its effectiveness in preventing unauthorised data access and encryption.

5. Modelling BBSV in Coloured Petri Nets

In this section, we provide a statistical analysis for the BBSV algorithm, outlining a secure communication protocol, modelled using Coloured Petri Nets (CPN). CPNs have been extensively employed in numerous studies to validate the proposition that vulnerabilities of a system or device can be exploited by adversaries [36]. This validation is achieved through the creation and simulation of comprehensive CPN models. This system is designed to facilitate the secure interaction of users with a TPM and a proxy server, incorporating procedures for user authentication, key distribution, and data encryption/decryption. The CPN model (Fig. 3) serves as a graphical and mathematical apparatus used in modelling intricate systems exhibiting concurrent behaviours. This model has been instrumental in the generation of the statistical analysis presented here. Furthermore, we furnish the outcomes pertaining to the BBSV key generation and key distribution, as rendered in the CPN context.

Table 2
State-space properties.

Property	Value
Nodes	29
Arcs	38
Secs	0
Status	Full

Table 3
Best integer bounds.

Property	Bounds	
	Upper	Lower
TPM′Application 1	1	0
TPM′DataStored 1	1	0
TPM′Data_signed 1	1	0
TPM′END 1	0	0
TPM′End 1	1	0
TPM′′Fresh 1	1	0
TPM′Key_Generation 1	1	0
TPM′Key_Pairs_Distribution 1	1	0
TPM′Proxy 1	1	0
TPM′Rk 1	1	0
TPM′SMS 1	1	0
TPM′TPM 1	1	0
TPM′Tags_Validation 1	1	0
TPM′User 1	1	0
TPM′encrypt_embedding_Tags 1	1	0
TPM′tags_removed 1	1	0
TPM′validation 1	1	0

5.1. State-space analysis

The state-space analysis of the BBSV model, as generated by CPN Tools, reveals several crucial aspects of the system's design and functionality, particularly in the context of security and reliability. Firstly, the state-space report indicates a comprehensive exploration of the model, with 29 nodes and 38 arcs (Table 2). The status *Full* suggests that all possible states and transitions have been thoroughly explored, ensuring that the model is complete and representative of all potential behaviours of BBSV. This thoroughness is essential for validating the integrity and robustness of the BBSV model, especially as it deals with critical aspects such as key generation, data storage, and validation.

The boundedness properties further strengthen the model's reliability. The upper and lower bounds for various processes, such as TPM′Data_signed, TPM′Application, and TPM′DataStored, indicate the system's operation within defined limits. This is critical in a security context, as it ensures predictable behaviour and prevents scenarios like buffer overflows or resource exhaustion, which could be exploited. The best upper multi-set bounds show the maximum capacity of each process, providing insight into the system's handling of different states and resources at peak levels (Table 3).

In terms of liveness properties, the report identifies dead markings and transition instances, such as TPM′Validated_False 1 (Table 4). This suggests that certain conditions or states leading to the end of a process prevent endless loops or deadlocks. The absence of live transitions means that the system will not enter into an endless cycle of operations, ensuring that it remains responsive and efficient.

Lastly, the fairness properties, indicating no infinite occurrence sequences, assure that the system is designed to prevent the monopolisation of resources or processes (Table 5). This is crucial for maintaining a balanced operation, where no single process or resource can dominate, leading to a more stable and equitable system behaviour. The state-space report of the TPM model showcases a well-constructed system with defined boundaries and predictable behaviour. These attributes are essential for a trusted security module, as they ensure not only the security but also the reliability and stability of the system in handling sensitive operations.

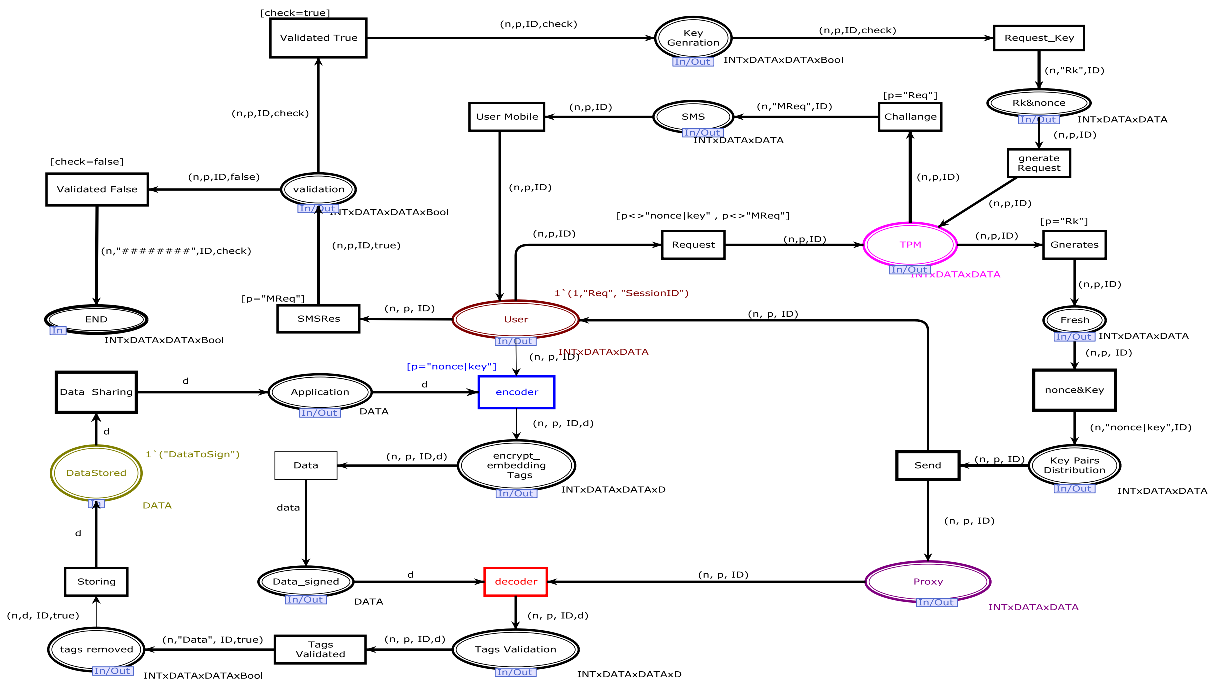


Fig. 3. The CPN model integrates user authentication, key distribution, and Data encryption/decryption for seamless and safe communication with trusted platform modules and proxy servers. (https://youtu.be/7UV2cwQSi8I).

Table 4
Liveness properties.

Property	Value
Dead markings	[1]
Dead transition instances	TPM'Validated_False 1
Live transition instances	None

Table 5
Fairness properties.

Property	Value
Infinite occurrence sequences	No

6. Experiment setup and evaluation

Our investigation aimed to assess the efficiency and robustness of the BBSV approach in defending against various ransomware attacks. The experimental framework consisted of three components: an endpoint; a proxy gateway; and a cloud-storage system.

The endpoint was simulated using an isolated virtual machine running Windows 10. The primary data source was the NYU fastMRI Initiative's medical-imaging dataset [37]. This dataset included over 1500 fully sampled knee MRIs and DICOM images from 10,000 clinical knee MRIs, which were transferred using the SMB protocol. To ensure data authenticity, distinctive digital signatures were embedded at the userspace level through an encoder before the data were committed to storage.

Our experimental setup was powered by an AMD Ryzen Thread-Ripper Pro 5975WX 32-core processor, equipped with 128 GB of RAM and a 1 TB SSD. This hardware configuration, running on Windows 11, provided substantial processing power and storage capacity for handling extensive datasets and complex models. For software, we used Python 3.10 and the Spyder integrated development environment (IDE) for code development and experimentation. Additionally, we used VMware virtual machines to accommodate the Proxy Gateway (running Ubuntu 24.04) and the endpoint running Windows 10.

The data were then transmitted to the proxy gateway, a virtual machine operating on Ubuntu 24.04. This gateway intercepted the data buffers and verified the embedded tags. At this gateway, the FUSE became operational, mounting the incoming data from the cloud onto a designated directory using the SMB protocol. The data were shared with the endpoint through the SMB protocol. Simultaneously, a decoder extracted and verified the embedded digital signatures. If the data signatures were successfully validated, the data proceeded to the cloud storage (using Azure). Conversely, if any manipulation of the signatures was detected, the proxy gateway blocked the data from advancing to storage, thus preventing malicious encryption.

6.1. Experimental setup

Fig. 4 illustrates the data-flow architecture in which files from an endpoint underwent processing through an Ubuntu server. This server was responsible for verifying tags. If the verification process indicated that the tags were not valid, suggesting potential encryption or modification by unauthorised users or applications, the data was blocked from being written to cloud storage. This setup was designed to ensure the integrity and authenticity of the data being stored. It is important to note that the challenge system has been demonstrated as part of the CPN, thereby avoiding the inclusion of redundant information.

Fig. C.8 in Appendix C illustrates our experimental setup, showing the proxy gateway's process for validating tags and the subsequent flag-raising mechanism. Our experiments demonstrate that BBSV is effectively safeguarded against malicious encryption, ensuring the integrity and security of the data throughout its lifecycle.

6.2. BBSV attack-resilience analysis

The resilience of the BBSV approach was tested using 80 distinct ransomware variants extracted from 70 ransomware families. In addition, a safeguard against Memory Disclosure Attacks was integrated, with key distribution and challenges presented at the external node. To enhance the robustness of the approach, a nonce was newly generated for every encoder and user-challenge instance. The data flow, from the endpoint to cloud storage via the proxy gateway, embodied a secure channel for data exchange. The evaluation of the system's performance was multi-faceted, focusing on its capability to preempt unauthorised

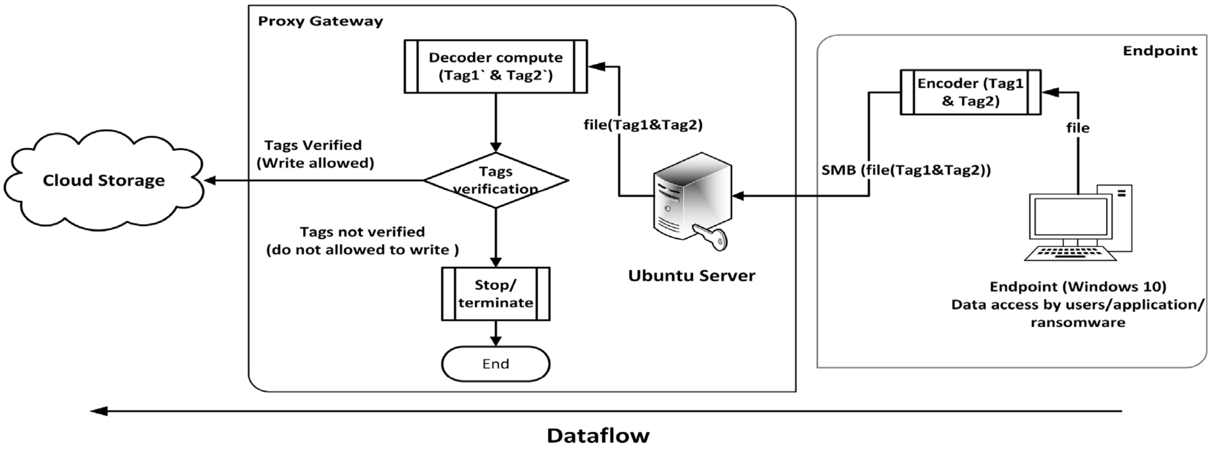
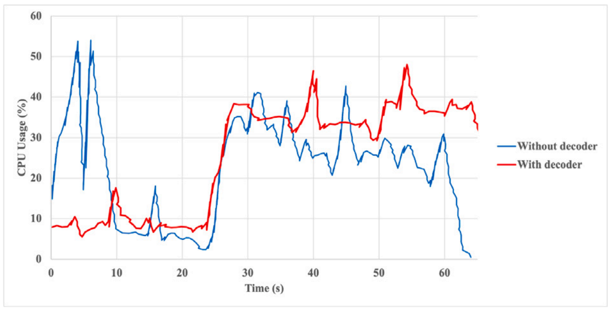
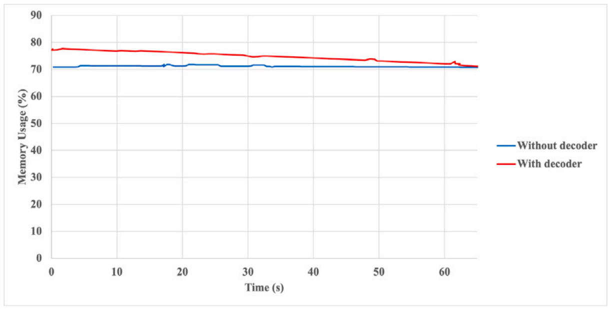

Proxy Gateway
Decoder compute (Tag1` & Tag2`)
file(Tag1&Tag2)
Tags Verified (Write allowed)
Tags verification
Cloud Storage
Tags not verified (do not allowed to write)
Stop/ terminate
End
Ubuntu Server
Endpoint
Encoder (Tag1 & Tag2)
file
SMB (file(Tag1&Tag2))
Endpoint (Windows 10) Data access by users/application/ ransomware
Dataflow

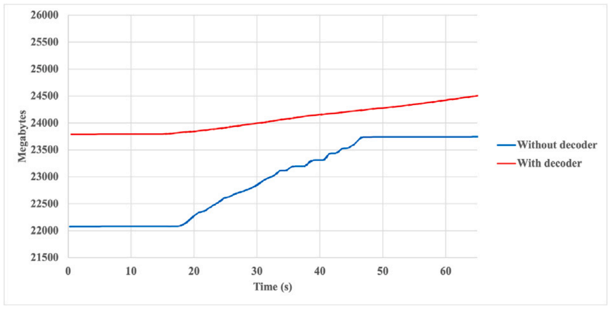
Fig. 4. BBSV data-flow architecture.



(a) CPU usage in copying files from an endpoint to cloud storage via Proxy Gateway.



(b) Memory usage in copying files from an endpoint to cloud storage via Proxy Gateway.



(c) Write usage in copying files from an endpoint to cloud storage via Proxy Gateway.

Fig. 5. Computational-cost analysis of BBSV decoder against Proxy Gateway without the use of a decoder.

data access, the efficiency of its data processing and verification, and its adeptness at handling diverse data types.

In the course of our experiments, we experienced no data loss (no false negatives) during the execution of the ransomware. However, a single data point was corrupted during the process of the ransomware loading data into memory and subsequently writing it back to storage. However, given that the actual data resided in the cloud, it remained recoverable. Moreover, we observed no instances of false positives, as the encoder successfully signed all the data, and the proxy was able to decode and verify these signatures reliably.

6.3. BBSV computational-cost analysis

In the course of our study, we conducted an extensive evaluation of the BBSV decoder's performance, focusing on its computational demands for CPU usage, memory consumption, and write overhead at the proxy gateway. The data used for this empirical analysis consisted of 1358 files of total size 1.8 GB. Our findings demonstrated an agreeable level of computational resources used during the operation of the proxy gateway. The test results yielded a detailed understanding of the functioning of the BBSV decoder, mapping its efficiency and effectiveness in terms of computational overhead. Fig. 5 shows the computational overhead associated with the proxy gateway, providing a visual representation of the decoder's performance metrics. The results in the figure offer comprehensive insights into the resource utilisation of the BBSV decoder and its impact on the system's overall performance compared with a proxy gateway without a decoder for the same period of the first 65 s. The experiments, both with and without the decoder, were conducted separately. However, the results are plotted together for comparison purposes.

6.4. Ransomware modus operandi against BBSV

Ransomware typically infiltrates a computational system, encrypts data of substantial value, and then demands a ransom in exchange for the decryption key. However, in the proposed scenario, the ransomware could adopt a more nuanced and sophisticated strategy. This might include importing files into the memory, transferring them to an external Command and Control (C2) server, replacing the local storage with encrypted files, and ultimately deleting the original files from the system. These processes can be conceptualised as a sequence of state transitions, starting with the transfer of a file from storage into the memory, followed by in-memory processing of the file, then exfiltrating it to the C2 server or rewriting the encrypted version back to a different location in storage, and finally, the removal of the original file.

In this section, we hypothetically develop possible attack scenarios in which ransomware developers might use unknown tactics, weaponry

$Wp(t')$, to circumvent the BBSV model. Assuming an adversary can bypass the BBSV model by launching an attack, the situation would evolve as follows. The adversary begins by loading files into the memory. They then transfer these files from their original position in the memory, such as a network-shared drive, using the SMB protocol, to a different location, provided the local storage has adequate capacity. By exploiting this covert route, the adversary could potentially transmit the exfiltrated data to a C2 server, thereby exposing a vulnerability in the BBSV model. Subsequently, they could then erase the original files from the network-shared drive, leaving no trace of their previous existence.

6.4.1. State-space and transition strategy to evade BBSV

We define an extended state space S, where each state $s \in S$ is a tuple: $s = (U, F, M, L, C2, E, D, K)$. Here, U represents the userspace, F the filesystem, M the memory, L the local storage, $C2$ the state of the C2 server, E the encoder, D the decoder, and K the key distribution and challenge system at the external node.

The initial state space $I \subseteq S$ represents the system's initial states. The transition relation $T \subseteq S \times S$ is defined such that $(s, s') \in T$ if and only if one of the following holds:
1. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $U' = \text{fetch}(F, SMB, M)$, representing the attacker fetching the file from the SMB and transferring it to memory.
2. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $L' = \text{write}(F, M, L)$, representing the attacker writing the file in memory to local storage.
3. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $M' = \text{read}(U, M)$, representing the attacker reading from memory.
4. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $C2' = \text{leak}(M, C2)$, representing the attacker leaking the file to a C2 server.
Here, *fetch*, *write*, *read*, and *leak* are functions representing the corresponding operations in the system.

The adversarial goal is defined using a predicate $G : S \to \{\text{true}, \text{false}\}$, such that $G(s) = \text{true}$ if and only if $s = s_{\text{bad}}$, where s_{bad} is a state in which the attacker has successfully written a file to another location in the local storage and leaked the file to a C2 server.

An adversarial attack is represented as a sequence of states $s_0, s_1, s_2, \ldots, s_n$ such that $s_0 \in I$, $(s_i, s_{i+1}) \in T$ for all $0 \le i < n$, and $G(s_n) = \text{true}$. This sequence signifies that the attacker started from an initial state, performed a sequence of fetch, write, read, and leak operations, and eventually reached the state where the file has been written to another location in the local storage and leaked to a C2 server.

6.4.2. Ransomware modus operandi: Leak and delete files

We further extend the state space S to include additional transitions:
1. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $F' = \text{delete}(F)$, representing the attacker deleting the original file from the system.
2. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $M' = \text{encrypt}(E, M)$, representing the attacker encrypting the file in memory.
Here, *delete* and *encrypt* are functions representing the corresponding operations in the system.

The adversarial goal can now be defined using a predicate $G : S \to \{\text{true}, \text{false}\}$, such that $G(s) = \text{true}$ if and only if $s = s_{\text{bad}}$, where s_{bad} is a state where the original file has been encrypted, written back to local storage, deleted, and leaked to a C2 server.

Therefore, an adversarial attack can be represented as a sequence of states $s_0, s_1, s_2, \ldots, s_n$ such that $s_0 \in I$, $(s_i, s_{i+1}) \in T$ for all $0 \le i < n$, and $G(s_n) = \text{true}$. This model represents an attacker starting from an initial state, fetching a file, encrypting it, writing the encrypted file to local storage, reading from memory, leaking the original file to a C2 server, deleting the original file, and eventually reaching the adversarial goal.

6.4.3. Mitigating adversarial attacks

We redefine the *fetch* and *delete* operations to include user authentication. Denote the authentication function as $\text{auth}(U)$. If $\text{auth}(U) = \text{true}$, the user is authenticated; if $\text{auth}(U) = \text{false}$, the user is not authenticated.

The transition relation $T \subseteq S \times S$ is now defined such that $(s, s') \in T$ if and only if one of the following holds:
1. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $U' = \text{fetch}(F, SMB, M)$ if and only if $\text{auth}(U) = \text{true}$, representing the attacker fetching the file from the SMB and transferring it to memory only if the user is authenticated.
2. $s = (U, F, M, L, C2, E, D, K)$ and $s' = (U', F', M', L', C2', E', D', K')$, where $F' = \text{delete}(F)$ if and only if $\text{auth}(U) = \text{true}$, representing the attacker deleting the original file from the system only if the user is authenticated.

For the other state transitions, the conditions remain the same as before.

Now, the attacker must authenticate before fetching or deleting a file, adding an additional layer of security and mitigating the risk of unauthorised file access or deletion.

We further extend our adversarial goal to include a state in which the adversary attempts to perform unauthenticated *fetch* or *delete* operations. Specifically, we define $G(s) = \text{true}$ if and only if $s = s_{\text{bad}}$, where s_{bad} is a state in which either:

- The original file has been encrypted and written back to local storage, the original file has been deleted, and the original (unencrypted or encrypted) file has been leaked to a C2 server; or
- The attacker attempts to *fetch* a file or *delete* a file without proper authentication.

This model now represents an attacker starting from an initial state, attempting to fetch a file, encrypt it, write the encrypted file to local storage, read from memory, leak the original file to a C2 server, delete the original file, and eventually reach the adversarial goal. With the added mitigation strategy, any attempts to fetch or delete a file without proper authentication would be blocked, preventing the attacker from achieving their goal. This scenario depends on environment sensitivities and the principle of least privilege, which means that users or processes should only have the minimum permissions necessary to perform their tasks. By enforcing authentication and authorisation, we ensure that only those with a legitimate need can delete files, reducing the attack surface.

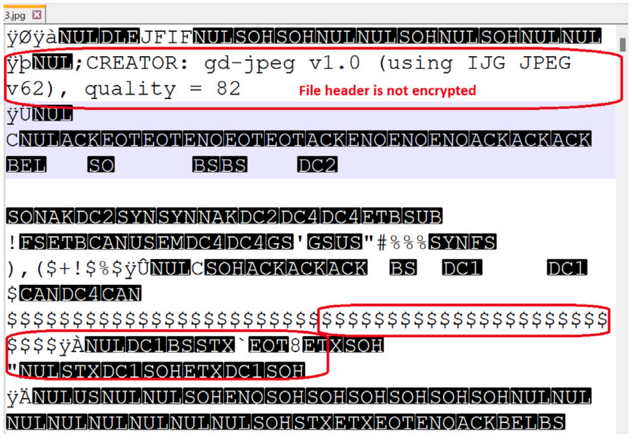
7. Comparing BBSV and ShieldFS

We compare our approach of BBSV with ShieldFS, which has been proposed for analysing filesystem I/O Request Packets to detect ransomware activity [19]. ShieldFS uses a methodical approach to ransomware detection and recovery, leveraging extensive large data collection to inform its protective mechanisms. ShieldFS's core data-gathering tool, IRPLogger,[2] was used to analyse filesystem interactions under normal and ransomware-compromised conditions on a variety of real-world systems. Through the data obtained from volunteers' machines and controlled ransomware infections in virtual environments, the authors created a large dataset (limited ransomware) that accurately represented the I/O patterns of both legitimate and malicious activities. These data served as the foundation for developing a detection system that used a copy-on-write mechanism to protect and recover files in the event of an attack. The method focused on the runtime behaviour of processes, using machine learning models to identify ransomware activity based on deviations from established benign patterns. ShieldFS components and functions are summarised in Table 6.

[2] https://github.com/pagiux/IRPLogger/tree/master.

Table 6
Summary of ShieldFS components and functions.

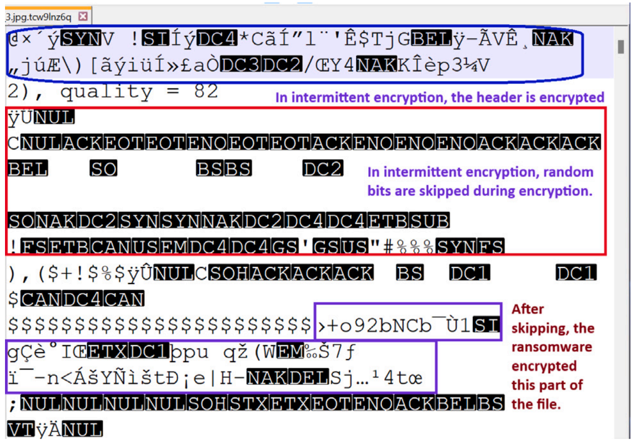
Component	Function
IRPLogger	Logs low-level I/O requests
Adaptive models	Machine learning models for anomaly detection
Copy-on-Write	Dynamic backups upon file modification
Process/System-centric Models	Holistic behaviour analysis for threat identification
CryptoFinder	Memory scan for encryption activity indicators
Transaction log	Records IRPs for file recovery
Whitelisting	Prioritises directories/processes for monitoring



```
ÿØÿàNULDLEJFIFNULSOHSOHNULNULSOHNULSOHNULNUL
ÿþNUL;CREATOR: gd-jpeg v1.0 (using IJG JPEG
v62), quality = 82
ÿÛNUL
CNULACKEOTEOTENQEOTEOTACKENQENQENQACKACKACK
BEL    SO        BSBS      DC2

SONAKDC2SYNSYNNAKDC2DC4DC4ETBSUB
!FSETBCANUSEMDC4DC4GS'GSUS"#%%%SYNFS
),($+!$%$ÿÛNULCSOHACKACKACK  BS  DC1      DC1
$CANDC4CAN
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$ÿÀNULDC1BSSTX`EOT8ETXSOH
"NULSTXDC1SOHETXDC1SOH
ÿÄNULUSNULNULSOHENQSOHSOHSOHSOHSOHSOHNULNUL
NULNULNULNULNULNULSOHSTXETXEOTENQACKBELBS
```

Fig. 6. Normal hex file of an image before BlackBasta encrypts the file.


3.jpg.tcw9lnz6q
In intermittent encryption, the header is encrypted
In intermittent encryption, random bits are skipped during encryption.
After skipping, the ransomware encrypted this part of the file.

```
@×´ýSYNV !SIÍýDC4*CãÍ”l¨'Ê$TjGBELÿ-ÃVÊ¸NAK
„júÆ\)[ãýiüÍ»£aÒDC3DC2/ŒY4NAKKÎèp3¼V
2), quality = 82
ÿÛNUL
CNULACKEOTEOTENQEOTEOTACKENQENQENQACKACKACK
BEL    SO        BSBS      DC2

SONAKDC2SYNSYNNAKDC2DC4DC4ETBSUB
!FSETBCANUSEMDC4DC4GS'GSUS"#%%%SYNFS
),($+!$%$ÿÛNULCSOHACKACKACK  BS  DC1      DC1
$CANDC4CAN
$$$$$$$$$$$$$$$$$$$$$$$$$›+o92bNCb¯Ù1SI
gÇè°IŒETXDC1þpu qž(WEM‰Š7ƒ
ï¯-n<ÁšYÑìštÐ¡e|H-NAKDELSj…¹4tœ
;NULNULNULNULSOHSTXETXEOTENQACKBELBS
VTÿÄNUL
```

Fig. 7. BlackBasta encrypts 64-byte blocks from the beginning of the file but with a reduced skip interval of 128 bytes between each encrypted segment.

7.1. ShieldFS challenges

ShieldFS uses adaptive models, including copy-on-write mechanisms, and process and system-centric detection models, which provide a foundation for defending against a range of ransomware tactics. However, ransomware is constantly evolving, with new methods being developed to bypass security measures. Two such tactics are *intermittent encryption* and *partial encryption*, which are designed to evade detection by deviating from typical behaviour patterns that security tools like ShieldFS are trained to detect. These tactics are dynamic and can be changed from one variant to the next.

Intermittent encryption refers to ransomware that encrypts files sporadically or at varying intervals, possibly avoiding generating a consistent pattern that machine learning models expect for detection. Partial encryption, on the other hand, encrypts only parts of files or certain types of files, which might not trigger the thresholds set for abnormal I/O or file-access rates that are indicative of ransomware. BlackBasta uses intermittent-encryption tactics to evade ShieldFS-like countermeasures. Figs. 6(before) and 7(after) illustrate files encrypted using the intermittent encryption method, which poses significant challenges to proposed methods that rely on file system heuristic calls and the frequency of file system attributes as indicators of encryption.

While ShieldFS's approach of a sophisticated detection algorithm that monitors for features indicative of ransomware and a transaction log to recover affected files is comprehensive, the system's efficacy against these new methods will depend on several factors.

- *Model Adaptability*: The ability of ShieldFS's machine learning models to adapt to new ransomware patterns based on the gathered data.
- *Feature Sensitivity*: The sensitivity of ShieldFS's detection features to pick up on subtle and sparse signs of encryption that occur with intermittent or partial encryption methods.
- *Behavioural Analysis*: The extent to which ShieldFS analyses the behaviour of processes over time, which could potentially identify ransomware that employs low and slow encryption strategies.
- *Thresholds and Heuristics*: The configuration of detection thresholds and heuristics in ShieldFS's models may need to balance being stringent enough to catch new ransomware techniques and flexible enough to avoid false positives.

Compared to ShieldFS, BBSV has been designed to enhance the effectiveness of systems that rely on behavioural analytics for monitoring filesystem activities. This protocol establishes a cryptographic signature for files at the time of creation or modification before the data are stored in cloud environments. The integration of such a mechanism enables the BBSV protocol to offer a strong defence against advanced adversarial tactics such as intermittent and partial encryption strategies. The implementation of BBSV ensures that any changes to the file block during file transfer result in an anomaly, which can then isolate the file for further investigation by administrators.

8. Conclusion

In this study, we introduced a novel approach called the Buffer-Based Signature Verification (BBSV) method. This method involves embedding 32-byte tags into files before they are stored in a cloud environment. The primary components of the BBSV include encoding and decoding submodules. The encoding submodule reads data, generates signatures, and encrypts the image before storing it in the cloud. The decoding submodule decrypts and validates digital signatures before any modification or encryption is applied to the data by an application. The BBSV's key generation uses a Trusted Platform Module (TPM), which allows the endpoint user to resist cryptographic attacks. The key generation and distribution process, together with the algorithm, were tested using Coloured Petri Nets, ensuring a robust and adaptable framework for the system. Our design enables secure interactions between users, a TPM, and a proxy server while incorporating comprehensive procedures for user authentication, key distribution, and data encryption/decryption. Our findings demonstrated the successful application of BBSV in protecting data against real-world ransomware attacks. In our experiments, the BBSV solution showed high efficiency and robustness, with no data loss or false negatives during ransomware executions. Only one data point was corrupted when the ransomware loaded and wrote back data. Our system reported zero false positives, and the encoding and decoding processes worked flawlessly, confirming the solution's effectiveness. Future research can explore optimising the encoding and decoding processes to reduce computational overhead and improve efficiency, particularly in resource-constrained environments, and investigate the adaptability of BBSV to WebDAV, AWS S3, and SFTP services. We are also focusing on the performance and scalability of the entire system using methods available in the literature [38], investigating the implementation of randomised sampling and incremental update techniques to mitigate computational bottlenecks. By exploring these strategies, we aim to enhance system efficiency and scalability, paving the way for more robust handling of continuous write scenarios in the BBSV model.

CRediT authorship contribution statement

Arash Mahboubi: Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Data curation, Conceptualization. **Seyit Camtepe:** Writing – review & editing, Writing – original draft, Validation, Investigation, Formal analysis, Conceptualization. **Keyvan Ansari:** Writing – review & editing, Writing – original draft, Visualization, Validation, Resources, Investigation, Conceptualization. **Marcin Pawłowski:** Writing – review & editing, Resources, Methodology, Investigation. **Paweł Morawiecki:** Writing – original draft, Methodology, Investigation, Formal analysis. **Hamed Aboutorab:** Visualization, Validation, Methodology, Investigation, Formal analysis. **Josef Pieprzyk:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision. **Jarek Duda:** Writing – original draft, Visualization, Validation, Supervision, Methodology.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs.

Data availability

No data was used for the research described in the article.

Acknowledgement

The work has been supported by the Cyber Security Research Centre Limited, whose activities are partially funded by the Australian Government's Cooperative Research Centre Program.

Appendix A. Boundedness properties

The boundedness properties further strengthen the model's reliability. The upper and lower bounds for various processes, such as TPM′Application, TPM′DataStored, and TPM′Data_signed, indicate the system's operation within defined limits (Table A.7). This is critical in a security context, as it ensures predictable behaviour and prevents scenarios like buffer overflows or resource exhaustion, which could be exploited. The best upper multi-set bounds show the maximum capacity of each process, providing insight into the system's handling of different states and resources at peak levels.

Appendix B. Comparison of security technologies

Hardware Security Module (HSM): HSMs offer a high-security level for cryptographic operations, featuring tamper-resistant design and centralised key management. However, they come with high costs, integration complexity, performance overhead, scalability limitations, vendor lock-in, and physical security requirements [39].

Intel Software Guard Extensions (SGX): SGX provides strong isolation of sensitive code and data, supports complex applications within enclaves, and has a growing ecosystem. Its limitations include limited memory size for enclaves, performance overhead due to context switches, a complex development process, limited I/O operations, susceptibility to side-channel attacks, and an evolving ecosystem [40].

ARM TrustZone: TrustZone offers a secure execution environment with wide support across ARM-based devices and low-performance overhead for many applications. Challenges include complex development and debugging, limited resources in the secure world, compatibility issues across different ARM systems, and dependence on hardware design [41].

Trusted Platform Module (TPM): TPMs are characterised by a hardware-based root of trust, secure key storage and management, support for secure boot and attestation, and widespread adoption and standardisation. On the downside, TPMs have limited storage capacity, performance overhead for cryptographic operations, complex integration, dependence on firmware and software, limited functionality, vendor-specific implementations, physical security risks, and key management complexity [42].

Table B.8 provides a comprehensive comparison of several prominent security technologies, highlighting their respective advantages and disadvantages. This comparison is essential for understanding the trade-offs involved in selecting the appropriate security solution for different applications.

Appendix C. Experimental setup

The experimental setup was established with data populated into Azure storage and an Azure virtual machine (Ubuntu) configured to serve as a proxy gateway. The endpoint (Windows 10, loaded with ransomware) facilitated the dissemination of cloud data through this proxy gateway (decoder). All uploaded data underwent a signing process with an encoder. This proxy gateway performed verification of Tag1 and Tag2 prior to any potential data modifications by users or ransomware. Fig. C.8 illustrates the proxy gateway's tag-validation process and subsequent flag-raising. Our findings demonstrate that BBSV effectively safeguards against unauthorised modifications.

Appendix D. Supplementary data

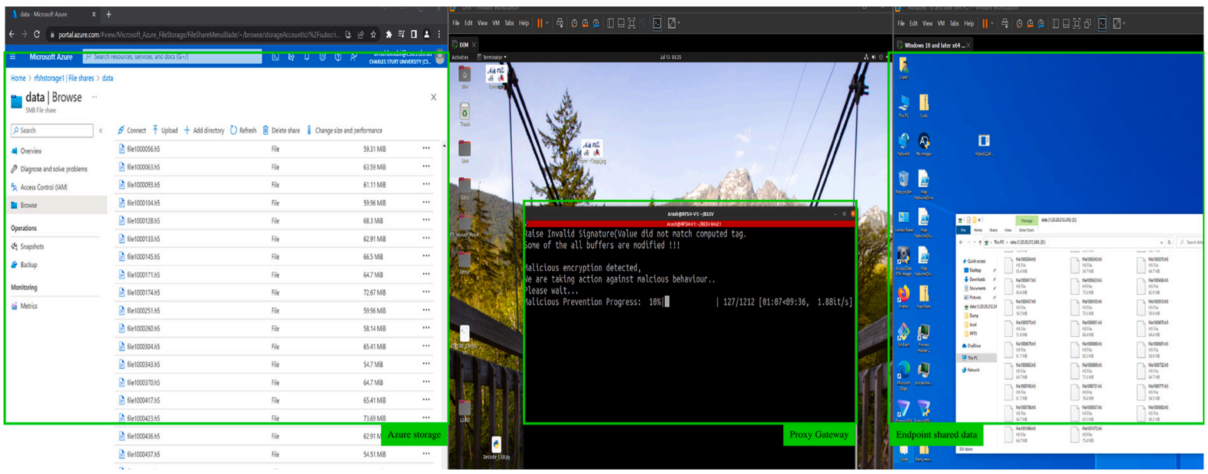
Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jisa.2024.103873.

Table A.7
Best upper multi-set bounds.

Property	Bound
TPM′Application 1	1\`"Data"++ 1\`"DataToSign"
TPM′DataStored 1	1\`"Data"++ 1\`"DataToSign"
TPM′Data_signed 1	1\`"DATA_tg1_Tag2"
TPM′END 1	empty
TPM′End 1	1\`"Data"
TPM′Fresh 1	1\`(1,"Rk","SessionID")
TPM′Key_Generation 1	1\`(1,"MReq","SessionID",true)
TPM′Key_Pairs_ Distribution 1	1\`(1,"nonce|key","SessionID")
TPM′Proxy 1	1\`(1,"nonce|key","SessionID")
TPM′Rk 1	1\`(1,"Rk","SessionID")
TPM′SMS 1	1\`(1,"MReq","SessionID")
TPM′TPM 1	1\`(1,"Req","SessionID")++ 1\`(1,"Rk","SessionID")
TPM′Tags_Validation 1	1\`(1,"nonce|key","SessionID", "DATA_tg1_Tag2")
TPM′User 1	1\`(1,"MReq","SessionID")++ 1\`(1,"Req","SessionID")++ 1\`(1,"nonce|key","SessionID")
TPM′encrypt_embedding_ Tags 1	1\`(1,"nonce|key","SessionID","DataToSign")
TPM′tags_removed 1	1\`(1,"Data","SessionID",true)
TPM′validation 1	1\`(1,"MReq","SessionID",true)

Table B.8
Comparison of security technologies.

Technology	Pros	Cons
Hardware Security Module (HSM)	• High-security level for cryptographic operations • Tamper-resistant design • Centralised key management	• High cost (initial and maintenance) • Integration complexity • Performance overhead • Scalability limitations • Vendor lock-in • Physical security requirements
Intel Software Guard Extensions (SGX)	• Strong isolation of sensitive code and data • Supports complex applications within enclaves • Growing ecosystem and support	• Limited memory size for enclaves • Performance overhead due to context switches • Complex development process • Limited I/O operations • Susceptible to side-channel attacks • Evolving ecosystem
ARM TrustZone	• Provides a secure execution environment • Wide support across ARM-based devices • Low-performance overhead for many applications	• Complex development and debugging • Limited resources in the secure world • Compatibility issues across different ARM systems • Dependence on hardware design
Trusted Platform Module (TPM)	• Hardware-based root of trust • Secure key storage and management • Supports secure boot and attestation • Widely adopted and standardised	• Limited storage capacity • Performance overhead for cryptographic operations • Complex integration • Dependence on firmware and software • Limited functionality • Vendor-specific implementations • Physical security risks • Key management complexity



file1000056.h5	File	59.31 MiB
file1000063.h5	File	63.59 MiB
file1000093.h5	File	61.11 MiB
file1000104.h5	File	59.96 MiB
file1000128.h5	File	68.3 MiB
file1000133.h5	File	62.91 MiB
file1000145.h5	File	66.5 MiB
file1000171.h5	File	64.7 MiB
file1000174.h5	File	72.67 MiB
file1000251.h5	File	59.96 MiB
file1000260.h5	File	58.14 MiB
file1000304.h5	File	65.41 MiB
file1000343.h5	File	54.7 MiB
file1000370.h5	File	64.7 MiB
file1000417.h5	File	65.41 MiB
file1000423.h5	File	73.69 MiB
file1000436.h5	File	62.91 M
file1000437.h5	File	54.51 MiB

```
Arash@RFSH-V1: ~/BSSV
Raise Invalid Signature(Value did not match computed tag.
Some of the all buffers are modified !!!

Malicious encryption detected,
We are taking action against malcious behaviour..
Please wait...
Malicious Prevention Progress:  10%|█           | 127/1212 [01:07<09:36,  1.88it/s]
```

Fig. C.8. Demonstration of the experimental setup.

References

[1] Noe Guillaume, Read Brendan, Vizza Tony. Cyber in 2023: Evolving threats and resilience. Gov Dir 2023;75(2):830–5.

[2] Davis Wes. MOVEit cyberattacks: keeping tabs on the biggest data theft of 2023. Verge 2023. URL https://www.theverge.com/23892245/moveit-cyberattacks-clop-ransomware-government-business.

[3] Toulas Bill. Rhysida ransomware wants $3.6 million for children's stolen data — bleepingcomputer.com. 2024, https://www.bleepingcomputer.com/news/security/rhysida-ransomware-wants-36-million-for-childrens-stolen-data/. [Accessed 05 June 2024].

[4] Henson Val, van de Ven Arjan, Gud Amit, Brown Zach. Chunkfs: Using divide-and-conquer to improve file system reliability and repair. In: Proceedings of the 2nd conference on hot topics in system dependability - volume 2. HOTDEP '06, USA: USENIX Association; 2006, p. 7.

[5] Cornell Brian, Dinda Peter A, Bustamante Fabián E. Wayback: A user-level versioning file system for linux. In: Proceedings of usenix annual technical conference, FREENIX track. 2004, p. 19–28.

[6] Ungureanu Cristian, Atkin Benjamin, Aranya Akshat, Gokhale Salil, Rago Stephen, Całkowski Grzegorz, Dubnicki Cezary, Bohra Aniruddha. HydraFS: A High-Throughput file system for the HYDRAstor Content-Addressable storage system. In: 8th USeNIX conference on file and storage technologies (FAST 10). San Jose, CA: USENIX Association; 2010.

[7] Mahboubi Arash, Ansari Keyvan, Camtepe Seyit, Duda Jarek, Morawiecki Paweł, Pawłowski Marcin, Pieprzyk Josef. Digital immunity module: Preventing unwanted encryption using source coding. 2022, http://dx.doi.org/10.36227/techrxiv.17789735.v1.

[8] Huai Qianbo, Hsu Windsor, Lu Jiwei, Liang Hao, Xu Haobo, Chen Wei. XFUSE: An infrastructure for running filesystem services in user space. In: 2021 USeNIX annual technical conference. USeNIX ATC 21, USENIX Association; 2021, p. 863–75, URL https://www.usenix.org/conference/atc21/presentation/hsu.

[9] Mahboubi A, Camtepe S, Morarji H. A study on formal methods to generalize heterogeneous mobile malware propagation and their impacts. IEEE Access 2017;5:27740–56. http://dx.doi.org/10.1109/ACCESS.2017.2772787.

[10] Sgandurra Daniele, Muñoz-González Luis, Mohsen Rabih, Lupu Emil C. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. 2016, arXiv preprint arXiv:1609.03020.

[11] Vinayakumar R, Soman K P, Senthil Velan K K, Ganorkar S. Evaluating shallow and deep networks for ransomware detection and classification. In: 2017 international conference on advances in computing, communications and informatics. ICACCI, 2017, p. 259–65. http://dx.doi.org/10.1109/ICACCI.2017.8125850.

[12] Zhang Hanqi, Xiao Xi, Mercaldo Francesco, Ni Shiguang, Martinelli Fabio, Sangaiah Arun Kumar. Classification of ransomware families with machine learning based on N-gram of opcodes. Future Gener Comput Syst 2019;90:211–21. http://dx.doi.org/10.1016/j.future.2018.07.052, URL http://www.sciencedirect.com/science/article/pii/S0167739X18307325.

[13] Aslan Ömer Aslan, Samet Refik. A comprehensive review on malware detection approaches. IEEE Access 2020;8:6249–71. http://dx.doi.org/10.1109/ACCESS.2019.2963724.

[14] Al-rimy Bander Ali Saleh, Maarof Mohd Aizaini, Shaid Syed Zainudeen Mohd. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. Comput Secur 2018;74:144–66. http://dx.doi.org/10.1016/j.cose.2018.01.001, URL https://www.sciencedirect.com/science/article/pii/S016740481830004X.

[15] Alhawi Omar MK, Baldwin James, Dehghantanha Ali. Leveraging machine learning techniques for windows ransomware network traffic detection. In: Dehghantanha Ali, Conti Mauro, Dargahi Tooska, editors. Cyber threat intelligence. Cham: Springer International Publishing; 2018, p. 93–106. http://dx.doi.org/10.1007/978-3-319-73951-9_5.

[16] Oz Harun, Aris Ahmet, Levi Albert, Uluagac A Selcuk. A survey on ransomware: Evolution, taxonomy, and defense solutions. ACM Comput Surv 2022;54(11s). http://dx.doi.org/10.1145/3514229.

[17] Ahmed Yahye Abukar, Koçer Barış, Huda Shamsul, Saleh Al-rimy Bander Ali, Hassan Mohammad Mehedi. A system call refinement-based enhanced Minimum Redundancy Maximum Relevance method for ransomware early detection. J Netw Comput Appl 2020;167:102753. http://dx.doi.org/10.1016/j.jnca.2020.102753, URL https://www.sciencedirect.com/science/article/pii/S1084804520302277.

[18] Kharaz Amin, Arshad Sajjad, Mulliner Collin, Robertson William, Kirda Engin. UNVEIL: A large-scale, automated approach to detecting ransomware. In: 25th USeNIX security symposium. USeNIX security 16, Austin, TX: USENIX Association; 2016, p. 757–72, URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz.

[19] Continella Andrea, Guagnelli Alessandro, Zingaro Giovanni, De Pasquale Giulio, Barenghi Alessandro, Zanero Stefano, Maggi Federico. ShieldFS: A self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd annual conference on computer security applications. ACSAC '16, New York, NY, USA: Association for Computing Machinery; 2016, p. 336–47. http://dx.doi.org/10.1145/2991079.2991110.

[20] Kharraz Amin, Kirda Engin. Redemption: Real-time protection against ransomware at end-hosts. In: Dacier Marc, Bailey Michael, Polychronakis Michalis, Antonakakis Manos, editors. Research in attacks, intrusions, and defenses. Cham: Springer International Publishing; 2017, p. 98–119.

[21] Baek Sungha, Jung Youngdon, Mohaisen David, Lee Sungjin, Nyang DaeHun. SSD-assisted ransomware detection and data recovery techniques. IEEE Trans Comput 2021;70(10):1762–76. http://dx.doi.org/10.1109/TC.2020.3011214.

[22] Morato Daniel, Berrueta Eduardo, Magaña Eduardo, Izal Mikel. Ransomware early detection by the analysis of file sharing traffic. J Netw Comput Appl 2018;124:14–32. http://dx.doi.org/10.1016/j.jnca.2018.09.013, URL https://www.sciencedirect.com/science/article/pii/S108480451830300X.

[23] Lee Seungkwang, su Jho Nam, Chung Doyoung, Kang Yousung, Kim Myungchul. Rcryptect: Real-time detection of cryptographic function in the user-space filesystem. Comput Secur 2022;112:102512. http://dx.doi.org/10.1016/j.cose.2021.102512, URL https://www.sciencedirect.com/science/article/pii/S0167404821003369.

[24] Aljabri Malak, Alhaidari Fahd, Albuainain Aminah, Alrashidi Samiyah, Alansari Jana, Alqahtani Wasmiyah, Alshaya Jana. Ransomware detection based on machine learning using memory features. Egypt Inform J 2024;25:100445. http://dx.doi.org/10.1016/j.eij.2024.100445, URL https://www.sciencedirect.com/science/article/pii/S1110866524000082.

[25] Chen Li, Yang Chih-Yuan, Paul Anindya, Sahita Ravi. Towards resilient machine learning for ransomware detection. 2018, CoRR arXiv:1812.09400, arXiv:1812.09400. URL http://arxiv.org/abs/1812.09400.

[26] von der Assen Jan, Feng Chao, Celdrán Alberto Huertas, Oleš Róbert, Bovet Gérôme, Stiller Burkhard. GuardFS: a file system for integrated detection and mitigation of linux-based ransomware. 2024, arXiv preprint arXiv:2401.17917.

[27] Ganfure Gaddisa Olani, Wu Chun-Feng, Chang Yuan-Hao, Shih Wei-Kuan. DeepWare: Imaging performance counters with deep learning to detect ransomware. IEEE Trans Comput 2023;72(3):600–13. http://dx.doi.org/10.1109/TC.2022.3173149.

[28] Mofidi Farhad, Hounsinou Sena G, Bloom Gedare. L-IDS: A multi-layered approach to ransomware detection in IoT. In: 2024 IEEE 14th annual computing and communication workshop and conference. CCWC, 2024, p. 0387–96. http://dx.doi.org/10.1109/CCWC60891.2024.10427870.

[29] Oz Harun, Aris Ahmet, Acar Abbas, Tuncay Güliz Seray, Babun Leonardo, Uluagac Selcuk. RøB: Ransomware over modern web browsers. In: 32nd USeNIX security symposium. USeNIX security 23, Anaheim, CA: USENIX Association; 2023, p. 7073–90, URL https://www.usenix.org/conference/usenixsecurity23/presentation/oz.

[30] Schmitt Paul, Iyengar Jana, Wood Christopher, Raghavan Barath. The decoupling principle: A practical privacy framework. In: Proceedings of the 21st ACM workshop on hot topics in networks. HotNets '22, New York, NY, USA: Association for Computing Machinery; 2022, p. 213–20. http://dx.doi.org/10.1145/3563766.3564112.

[31] Mercadier Darius, Dagand Pierre-Évariste. Usuba: high-throughput and constant-time ciphers, by construction. In: Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation. PLDI 2019, New York, NY, USA: Association for Computing Machinery; 2019, p. 157–73. http://dx.doi.org/10.1145/3314221.3314636.

[32] Bellare Mihir, Namprempre Chanathip. Authenticated encryption: Relations among notions and&nbsp;analysis of the generic composition paradigm. J Cryptology 2008;21(4):469–91. http://dx.doi.org/10.1007/s00145-008-9026-x.

[33] Bernstein Daniel J, et al. ChaCha, a variant of salsa20. In: Workshop record of SASC. Vol. 8, 2008, p. 3–5.

[34] Bernstein Daniel J. The poly1305-AES message-authentication code. In: International workshop on fast software encryption. Springer; 2005, p. 32–49.

[35] Aumasson Jean-Philippe, Fischer Simon, Khazaei Shahram, Meier Willi, Rechberger Christian. New features of latin dances: analysis of salsa, ChaCha, and rumba. In: International workshop on fast software encryption. Springer; 2008, p. 470–88.

[36] Norta Alex, Matulevičius Raimundas, Leiding Benjamin. Safeguarding a formalized blockchain-enabled identity-authentication protocol by applying security risk-oriented patterns. Comput Secur 2019;86:253–69. http://dx.doi.org/10.1016/j.cose.2019.05.017, URL https://www.sciencedirect.com/science/article/pii/S0167404818302670.

[37] Knoll Florian, Zbontar Jure, Sriram Anuroop, Muckley Matthew J, Bruno Mary, Defazio Aaron, Parente Marc, Geras Krzysztof J, Katsnelson Joe, Chandarana Hersh, Zhang Zizhao, Drozdzalv Michal, Romero Adriana, Rabbat Michael, Vincent Pascal, Pinkerton James, Wang Duo, Yakubova Nafissa, Owens Erich, Zitnick C Lawrence, Recht Michael P, Sodickson Daniel K, Lui Yvonne W. fastMRI: A publicly available raw k-space and DICOM dataset of knee images for accelerated MR image reconstruction using machine learning. Radiol: Artif Intell 2020;2(1):e190007. http://dx.doi.org/10.1148/ryai.2020190007, arXiv:https://doi.org/10.1148/ryai.2020190007. PMID: 32076662.

[38] Li Meng, Ding Hanni, Wang Qing, Zhang Mingwei, Meng Weizhi, Zhu Liehuang, Zhang Zijian, Lin Xiaodong. Decentralized threshold signatures with dynamically private accountability. IEEE Trans Inf Forensics Secur 2024;19:2217–30. http://dx.doi.org/10.1109/TIFS.2023.3347968.

[39] Mavrovouniotis Stathis, Ganley Mick. Hardware security modules. In: Secure smart embedded devices, platforms and applications. Springer; 2013, p. 383–405.

[40] Schunter Matthias. Intel software guard extensions: Introduction and open research challenges. In: Proceedings of the 2016 ACM workshop on software protection. 2016, p. 1–1.

[41] Pinto Sandro, Santos Nuno. Demystifying arm trustzone: A comprehensive survey. ACM Comput Surv (CSUR) 2019;51(6):1–36.

[42] Segall Ariel. Trusted platform modules: Why, when and how to use them. Institution of Engineering and Technology; 2016.