Next Generation Cryptographic Ransomware

Ziya Alper Genç(✉), Gabriele Lenzini, and Peter Y. A. Ryan

Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg
{ziya.genc,gabriele.lenzini,peter.ryan}@uni.lu

Abstract. We are assisting at an evolution in the ecosystem of cryptoware —the malware that encrypts files and makes them unavailable unless the victim pays up. New variants are taking the place once dominated by older versions; incident reports suggest that forthcoming ransomware will be more sophisticated, disruptive, and targeted. Can we anticipate how such future generations of ransomware will work in order to start planning on how to stop them? We argue that among them there will be some which will try to defeat current anti-ransomware; thus, we can speculate over their working principle by studying the weak points in the strategies that seven of the most advanced anti-ransomware are currently implementing. We support our speculations with experiments, proving at the same time that those weak points are in fact vulnerabilities and that the future ransomware that we have imagined can be effective.

Keywords: Software security and malware · Ransomware · Anti-Ransomware · Cryptographic Techniques · Security evaluation and measurement

1 Introduction

Cryptographic ransomware, a breed of malware (also known as cryptoware) that encrypts files, makes them inaccessible, and asks for a ransom to decrypt them —an action that victims are unable to do, if encryption is strong— has boomed in the last years. Their attacks have left in disarray companies and single users alike creating an economic damage that has been estimated at billions of US dollars [27].

As other virulent cyber-threats, ransomware evolves with time. In its latest 2018 annual incident report [29] Symantec shows that in the last two years about one hundred more new families of ransomware have emerged (although less in 2017 than in 2016) and that, although certain families representatives like Cerber, Locky, and TorrentLocker "have disappeared from the scene over the course of the year" (ibid) the number of new variants per families has increased by 46% in 2017, adding to the existing cryptoware samples about 350 new mutants.

Together with other white papers written by security professionals, like the report of Kaspersky [13] and that of Barkly [1], such studies present ransomware as a malware stock in continuous evolution, "a lucrative venture for cybercriminals, spurring an increase in ransomware variants and their sophistication" (ibid).

In the attempt to contain the damage, anti-malware research has reacted promptly. Anti-ransomware applications have stopped a tantamount number of attacks (5.4 billions of them from some `WannaCry` variants only [1]) but defenders and attackers are embraced in a race that has just started. At today, the severity of ransomware threat is increasing and the worst is yet to come.

Is there a way to stop the threat? Is there a way to anticipate how the future generation of ransomware will look like? Although at the time of writing, statistics report that the sheer numbers of attacks is slowing down —Barkly in its blog[1] says that "in order to pull off a successful ransomware heist, the stars really have to align for attackers. Not only do they have to infect a victim who doesn't have reliable backups (or the time/resources required to use them), the victim also has to have quick and easy access to cryptocurrency, and be willing to put their trust in a criminal and pay them upfront. Making matters more difficult, attackers also have to price their ransom demands just right."— ransomware are expected to become more sophisticated and more disruptive [1]. Those implementing strong cryptography like ExPetr, Petya and NotPetya are even being used as disk wipers, that is, have become weapons of digital destruction in waster operations of cyberwar[2].

Thus, the research question for us researchers is whether there is a way to anticipate into what those sophisticated cryptoware will evolve in such a way to be prepared when the attacks will come. There is of course a great amount of criminal strategies that could work. Ransomware engineers can be quite inventive in this business. However, in our opinion, there is at least one direction that future ransomware will take, and we can guess it without invoking any foresight skill. If the history of malware and virus teaches us something (*e.g.,* see [30,11]), some new generation of the threat will be designed to respond to existing protections. Thus assuming that in the forthcoming generations of ransomware there will be some trying to overcome those protections, we can study the weaknesses in these latter's working principles and imagine what those evasive ransomware can do to dribble some of the most modern anti-ransomware strategies.

The exercise is not exempt from ethical consequences. As J. P. Sullins points out in "it must be acknowledged that working with malware is not ethically neutral" [28]. We discuss our position in this regards in §7.1. The paper opens with a review of seven of the most advanced anti-ransomware strategies (§2). Then it discusses their limitations (§2), and speculates on what a ransomware can do to evade their guard (§3 and §4). To prove that our speculation are in fact more than a thought experiment, we implemented the ransomware samples we have imagined and prove that it actually pass untouched the anti-ransomware applications, if they are available to us (§5 and §6). For those whose code is closed, or not yet implemented (*e.g.,* only described in research papers) we argue how our implementation is able to overcome them. We conclude the paper by

[1] Barkly, Must-Know Ransomware Statistics 2018, https://blog.barkly.com/ransomware-statistics-2018.

[2] For this reason, some does not even consider them be ransomware; they are however cryptoware, and therefore in the scope of this paper's research.

pointing the future work and by discussing the ethical choices that we had to take in this kind of research and our motivation to even start such work, and the code of conduct that we commit ourselves to follow (§7).

2 Defense Techniques: The State of the Art

Cryptographic ransomware families share a common goal: to encrypt a victim's files. They also share a few fundamental tasks that they necessarily have to execute to achieve the goal. For instance, they have to manage encryption and decryption keys; and they have to read, encrypt (and if the victim is lucky) decrypt, and write files. However, cryptographic ransomware comes in different forms. Although constrained to perform those common steps, they can reach the goal in different ways, so giving raise to different families of them.

For the same reason there are also many potential, not all necessarily effective, strategies to counteract ransomware. Current anti-ransomware approaches implement mainly two strategies: *key-oriented protection* and *behavioral analysis*.

Key-oriented Protection (KP). The rationale of those who follow a key-oriented protection strategy is that ransomware needs encryption keys and therefore it is better to keep those keys under control. "Keep keys under control" is not a simple action; current solutions have interpreted and implemented it in at least three distinguished methods:

(**KP**-i) - *controlling accesses over random number generators.* In this method the access to Cryptographically Secure Pseudo Random Number Generators (CSPRNGs) is controlled. CSPRNGs are functions that return good quality random numbers, which are essential ingredient to construct strong encryption keys. UShallNotPass [10] uses this principle. It allows access to CSPRNGs only if the call comes from a whitelisted application; all unauthorized processes are blocked and the callers are terminated.

(**KP**-ii) - *placing backdoors in random number generators.* In this strategy, a trapdoor is inserted to the CSPRNG of the host system. The aim of this trapdoor is to enable reproducing the previous outputs of CSPRNG for a given time. Thus, the random numbers used by ransomware as a seed can be obtained after an attack. Using these seed values, the keys used by ransomware are re-derived and the files are restored. In [16], Kim *et al.* proposed this technique to mitigate ransomware.

(**KP**-iii) - *escrowing encryption keys.* In this approach, cryptographic Application Programming Interfaces (APIs) are hooked, encryption keys and other parameters are acquired, and stored in a secure location. After a ransomware incident, these materials are used to recover the files. The first key-escrow based ransomware defense systems are proposed independently by Lee *et al.* [18] and Palisse *et al.* [21] and focused on only the built-in cryptographic APIs. Later, PayBreak [17] extended this technique to include the functions in third-party cryptographic libraries.

Behavioural Analysis (BA). Defenses that implement behavior analysis, monitor the interactions of applications and measure certain factors that may indicate the presence of a ransomware activity. Solutions diversify depending on the indicators used to monitor for the presence of ransomware. We recognize four major methods:

(**BA**-i) - *measuring entropy inflation.* Encryption increases the entropy of the files. Therefore, encryption can be detected by measuring the entropy of files, before and after file-write operations. A rough estimate of the entropy e, of a byte array $(x_i)_{i=1}^{n}$ that is often used is Eq. (1).

$$e = \sum_{k=0}^{255} P_k^x \log_2 \frac{1}{P_k^x} \quad \text{where} \quad P_k^x = \frac{|\{i : x_i = k\}|}{n} \tag{1}$$

Monitoring entropy changes is a method commonly used by CryptoDrop [25], ShieldFS [2], UNVEIL [14] and Redemption [15].

(**BA**-ii) - *detecting content modification.* Modern cryptographic algorithms produce ciphertext that completely differs from the plaintext data. Therefore, if the similarity between original file and modified file is small, the file might have been encrypted. In this respect, CryptoDrop utilizes `sdhash` [23] tool to compute dissimilarity of files to detect encryption performed by ransomware.

(**BA**-iii) - *identifying file-type changes.* File type can be identified by position-sensitive tests, *e.g.,* reading byte values at specific locations in a file. In contrary to benign applications, ransomware changes this information when encrypting a file, transforming the file into an unknown type. Therefore, changing file types is a strong indicator of ransomware activity. For example, CryptoDrop uses `file` [4] utility to detect modifications of file types.

(**BA**-iv) - *testing goodness-of-fit.* Encryption produces data which have a pseudo-random distribution. Based on this fact, DaD [20] employs χ^2 goodness-of-fit test to determine if the written data is close to random distribution and conclude that the file is being encrypted. To this aim, observed byte array is put into a frequency histogram with class interval 1 from 0 to 255. Let N_i denote the number of variates in bin i, and n_i be a known distribution. The χ^2 test value of this array is computed as in Eq. (2)

$$\chi^2 = \sum_i \frac{(N_i - n_i)^2}{n_i} \tag{2}$$

Indicators do recognize ransomware activities but also benign applications, *e.g.,* file compression utilities, show similar patterns. False positives can be reduced by combining indicators, as CryptoDrop does with indicators (**BA**-i), (**BA**-ii) and (**BA**-iii).

There are other indicators based on file access patterns *e.g.,* read/write/delete operations, access frequency, observed in ransomware attacks. Redemption, ShieldFS and UNVEIL use these indicators, but these systems are left for a future analysis. The analyzed systems in this paper and their corresponding defense techniques are given in Table 1.

Table 1. Select anti-ransomware systems and their main defense methods.

System	(**KP**-i)	(**KP**-ii)	(**KP**-iii)	(**BA**-i)	(**BA**-ii)	(**BA**-iii)	(**BA**-iv)
Kim *et al.* [16]		•					
CryptoDrop [25]				•	•	•	
Lee *et al.* [18]			•				
UShallNotPass [10]	•						
Palisse *et al.* [21]			•				
DaD [20]							•
PayBreak [17]			•				

3 Vulnerability Analysis of Countermeasures

"Every law has a loophole" says an old proverb, meaning that once a rule is known, it becomes known also how to evade it. This holds true also in the ransomware *versus* anti-ransomware arms race and in both ways. Knowing how ransomware works, one can design more effective defenses; knowing how defenses work, one can design more penetrating ransomware. In this section we discuss potential limitations in current anti-ransomware, and we imagine and discuss how future generation ransomware could evolve to overcome those defenses. In this exercise, we apparently take the side of ransomware but the goal is to stimulate the scientific community to anticipate better defenses that can work not only against current ransomware but also against forthcoming generation of them. This choice is not exempt from consequences. We discuss in §7 the ethical aspects in this research and we comment on the code of conduct we have committed ourselves to in developing this work.

3.1 Limits of Key-Oriented Protection

Key-oriented protection defenses aim at to prevent ransomware from using, undisturbed, cryptographic APIs.

In this respect, (**KP**-i) controls CSPRNG APIs on the host system, and (**KP**-ii) inserts a backdoor into CSPRNG APIs. A ransomware may evade these defences by using an alternative source of randomness. The critical question is whether there exist sources of randomness that are as good as CSPRNGs. We will elaborate more on this approach in §4.

Instead, (**KP**-iii) logs parameters and outputs of CSPRNG, built-in cryptographic APIs, and *recognized* functions in third-party libraries. As stated in [17], the critical limitation of this approach is that recognizing statically linked functions from third-party libraries is sensitive to *obfuscation*. Obfuscation does not affect recognizing calls to built-in APIs, so evasion is possible when ransomware binary is obfuscated and the ransomware refrain from using built-in APIs.

3.2 Limits of Behavioural Analysis

To detect cryptographic activities, behavioural analysis uses indicators, which are features revealing the presence of certain suspect behaviours; it also relies on constantly applying measurements and tests on files, before and after I/O operations.

In this respect, (**BA**-i) tests if the entropy of the file increases during a write operation using Eq. (1). It assumes that the encryption always increases the Shannon Entropy of a file. Indeed, this assumption holds for standard ciphers such as AES [3]. The entropy inflation test can be bypassed by changing the encryption algorithm with a one that preserves the entropy of the blocks.

Likewise, (**BA**-ii) compares the contents of a file before and after a file write operation and checks if the similarity score is above a threshold. A fully encrypted file should look like a random data and the comparison should yield a score close to 0, indicating a strong dissimilarity. This is true if the whole file is encrypted. A partially encrypted file, when compared with the plaintext version, is likely to result in high similarity scores: (**BA**-ii) may not be triggered while the file becomes practically unusable.

(**BA**-iii) can also be easily bypassed. If ransomware saves the file header, *i.e.,* does not encrypt the lead bytes of the file, and encrypts the rest, than the output of probe for file-type remains same. It should be noted that this information is generic, *i.e.,* publicly available, therefore cannot be considered as a critical data. Consequently, ransomware would not lose any profit by omitting the file-type identifying bytes. To nullify this strategy, anti-ransomware systems may utilize context-sensitive tests which scan entire file to detect a file's type, with the expense of degraded performance. In the experiments (§6), however, we haven't encountered such a detection. We remark that this defense might be bypassed by adding read/write routines for specific target file types, which is an implementation effort.

Finally, (**BA**-iv) tests if the written data is close to random distribution, based on the observation that standard ciphers like AES produce randomly distributed outputs. For this aim, χ^2 test given in Eq. (2) is used. However, if the χ^2 values can be kept constant during the obfuscation of file, this indicator will not trigger the alarm.

4 Future Ransomware Strategies

We present the blueprints of two novel ransomware samples that we claim are able to evade the defense systems listed in Table 1. The architecture of the samples is similar to that of WannaCry from the point of key management[3]. That is, each file in the victim's computer is encrypted with a unique symmetric key. Moreover, these symmetric keys are encrypted with a public key generated on the victim's

[3] This work focuses on the cryptographic aspects of ransomware. Other malicious operations, *e.g.,* spreading over network, are out of the scope of this paper.

computer. The corresponding secret key is then encrypted with the master public key embedded in the binary executable.

While this approach brings the risk of private key's being captured, it also removes the necessity of active connection to our hypothetical command and conquer (C&C) server which might be blocked by network firewalls and cause ransomware to fail.

4.1 Bypassing Key-Oriented Defenses

Our first construction targets key-oriented defense systems. As we point in §3.1, (**KP**-i) and (**KP**-ii) can be bypassed by utilizing an alternate randomness source. However, to defeat (**KP**-iii) completely, it is also required to statically link against a third-party library and apply obfuscation.

Deriving Encryption Keys A simple technique to generate the file encryption keys that malware might adopt is what is known in Cloud computing circles as *Convergent Encryption* [7]. Here, the cryptographic keys are derived from files themselves. A simple implementation is as follows. Let E be an encryption algorithm, H be a hash function, and F be the file. The technique consists in deriving the encryption key from hashing the file itself, that is $H(F)$. The resulting encryption is therefore $E(F, H(F))$.

The technique is free from the issues that may arise in the cloud computing. While convergent encryption is useful in certain scenarios, in the context of cloud computing, this technique may leak information as follows. For publicly-available plaintext files, the adversary can check and learn if the ciphertext belongs to these files. However, this is not really an issue in the context of ransomware: if the user still has the plaintext file(s), say in a backup, then the ransomware will not be effective anyway.

Our hypothetical ransomware thus computes $H(F)$ and derives the key by truncating this hash value to the length of K. This allows to evade the methods (**KP**-i) and (**KP**-ii). To win (**KP**-iii), we need a little more care: H and E must be statically-linked against a third-party library and obfuscated, otherwise (**KP**-iii) can acquire and store the result of H where K lies therein. The same requirement also applies to E. However, having a hash function in hand, the necessity of a block-cipher can also be fulfilled in the context of ransomware.

Symmetric Encryption Method Once the ransomware has got hold of good grade encryption keys then it can employ various well-established symmetric encryption techniques to the victim's files, for example a stream cipher, *e.g.,* based on a hash function in counter mode, or block cipher in an appropriate mode, *e.g.,* chained. The exact choice of algorithm is not so important as long as it sufficiently cryptographically strong to render cryptanalysis significantly more expensive than paying the ransom. However the algorithms should be fairly simple so as to be coded compactly and easy to obfuscate.

To encrypt the files we built a stream cipher using a keyed hash function build from H. Our construction utilizes H to generate a keystream in a similar way to the counter (CTR) mode of block ciphers. The keystream and the plaintext are combined using the *exlusive-or* (XOR) operation.

Let F be a plaintext stream such that $F = P_1 \,||\, P_2 \,||\, \ldots \,||\, P_n$ where each P_i has equal bit length to output of H, except possibly P_n, and $K = H(F)$. Encryption of F is done as follows:

$$S_i = H(K \,||\, i)$$
$$C_i = P_i \oplus H(K \,||\, S_i)$$

for $i = 1, 2, \ldots, n-1$. For $i = n$, $H(K \,||\, S_n)$ is truncated to the length of P_n.

In our design, we assume that H is (i) one-way: given K, it should be hard to find F such that $H(F) = K$; and (ii) collision-free: it should be hard to find $S_i \neq S_j$ such that $H(K \,||\, S_i) = H(K \,||\, S_j)$ (iii) pseudo-random: it is difficult to guess $H(K \,||\, i)$ —in our implementation, i has a fixed length of 32 bits— without knowing $K \,||\, i$.

Voiding Memory Dump Analysis Current software implementations of symmetric cryptographic algorithms require the encryption keys to be retrieved during the execution. Consequently, when encrypting files, the encryption keys reside in the memory area of the ransomware[4] process. Using this observation, defense techniques emerged (*e.g.,* [12]) which try to dump the memory of the encrypting process and extract the keys to roll-back the damage.

Deriving keys from the files' hashes overcomes this defense, as different files will result in distinct encryption keys. If a defense system detects files being encrypted, suspends the process and extracts the keys, it can only decrypt the file which is currently being accessed. Previous files cannot be recovered anymore as they are encrypted with different keys which were already destroyed at the time of detection.

File Based PRNG We have developed a pseudo-random number generator (PRNG) which inputs files, outputs pseudo-random bytes and provides the sufficient functionality for the purposes of ransomware. The PRNG has a pool, which is implemented as a byte array and initially filled with the hashes of files that will be encrypted. As the ransomware needs n bytes of pseudo-random number, n bytes are copied from the pool to the output buffer; the remaining bytes are shifted so that they will be in the next output. The output blocks are hashed and inserted again into the pool to prevent exhaustion. Our file based pseudo-random number generator (F-PRNG) is depicted in Fig. 1. It should be noted that as the files on victim's computer gets more exclusive, *i.e.,* different from other people's data, then the outputs of F-PRNG becomes harder to guess or reproduce after the attack as the plaintext versions of the files will be destroyed.

[4] Actually, ransomware might try to inject malicious code into other processes. In this case, memory of the encrypting process is dumped.

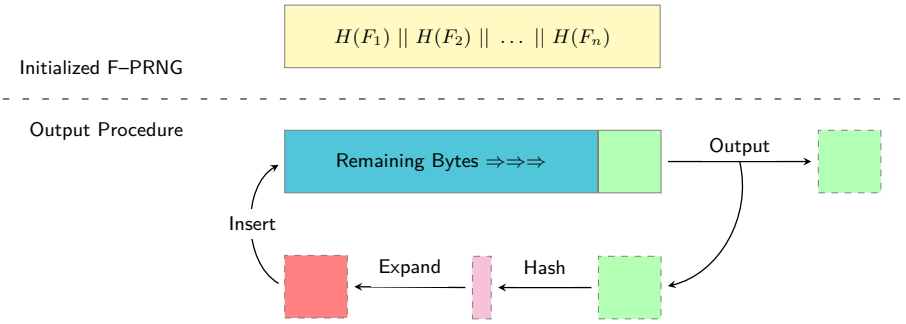

Fig. 1. Design view of the F-PRNG. The pool is seeded by file hashes. As pseudo-random bytes are requested from the F-PRNG, the output buffer is filled (dashed-green) with the requested amount. The remaining part of the pool (turquoise) is shifted accordingly. A copy of output bytes are hashed (purple), expanded (red) and inserted to the pool.

Expansion Process After providing the output bytes, that part is removed from the pool and the remaining bytes are shifted accordingly. This process shrinks the pool so that it exhausts in some finite time. To prevent this, we feed the pool with the pseudo-random numbers produced from the output that we call *expansion*. The method we use for expansion is similar to the approach used by Stark [26] and Eastlake [8], and described in Algorithm 1.

Algorithm 1 Expand a pseudo-random value to given length

```
1:  function EXPAND(input, n)
2:      global counter                          ▷ Pool keeps this counter
3:      ℓ ← Length(input)
4:      max ← ⌈n/ℓ⌉
5:      i = 0
6:      output = [ ]
7:      for i < max do
8:          counter += 1
9:          r = Hash(bytes || counter)          ▷ Generate pseudo-random chunk
10:         output = output || r                ▷ and add to output
11:     output = Truncate(output, n)            ▷ Output is truncated to n bytes
12:     return output
```

Asymmetric Key Pair Generation and Encryption Ransomware needs to store the locally generated file encryption keys securely. Modern ransomware employs asymmetric algorithms for this task.

Our imaginary ransomware also follows the same strategy. It employs the above F-PRNG to generate large primes to use in asymmetric algorithms, and to generate the padding values used for randomization of ciphertext.

4.2 Evading Behavioral Analysis

Our second ransomware targets behavioral based defense systems that constantly monitor file system activity and look for anomalies. In particular, its objective is to encrypt files without triggering the indicators described in §2.

The presented variant, rather than using standard block ciphers, basically employs a format preserving encryption algorithm. More specifically, the algorithm produces ciphertext which is a pure pseudo-random permutation of plaintext.

Bypassing File-Type Checks File-type probing is performed by inspecting the lead bytes of a file. Our ransomware therefore skips these bytes and starts encryption at a safe position. We identified this threshold empirically, testing over different file types including PDF, JPEG and DOCX. Our results shows that skipping the first 5120 bytes is sufficient for evading (**BA**-iii).

Preventing Dissimilarity Similarity of files is validated by comparing `sdhash` digests which produces a score between 0 and 100. According to the developers of `sdhash`, scores between 21-100 are considered as a strong indication of similarity [24]. In our experiments, comparing encrypted files with originals produces scores 0 or 1. However, we observed that partial encryption allows to obtain scores higher than 21, depending on the encryption ratio. (**BA**-ii) might set a lower threshold level, however, that would result in high false positive rates. Even in this case, tuning the encryption ratio would allow to keep this indicator silent. Fig. 2 shows the partially encrypted files of different types and their corresponding similarity scores.

Evading Statistical Tests (**BA**-i) measures the Shannon entropy of the files using Eq. (1), before and after file-write operations, and monitors the increase. Standard encryption algorithms usually dramatically increase the file-entropy and so this is detectable. Instead, one might use a transposition style cipher to obfuscate files: the ransomware generates a pseudo-random permutation of the bytes of the plaintext blocks. If, as is commonly the case, the anti-ransomware tools use the measure Eq. (1) then clearly permutation of the bytes leaves this invariant, and so this goes undetected.

There are two obvious drawbacks with this approach: firstly such a transposition encryption is cryptographically rather weak, and secondly it only works for this particular measure of entropy of a string. A weak encryption may be good enough for the purposes of the ransomware, as long as the cost of cryptanalysis exceeds the ransom. Given that an easy counter is to use a different measure of entropy, or better still use more than one, this would not seem to be a long-term viable solution for the writers of ransomware.

Lastly, pure permutation technique also works against (**BA**-iv), the single indicator that DaD employs to detect encryption. DaD computes the sliding median of the χ^2 values of the last fifty write operations and compares this result to the threshold level $\alpha_{RW} = 0.05$. However, the χ^2 statistics (computed

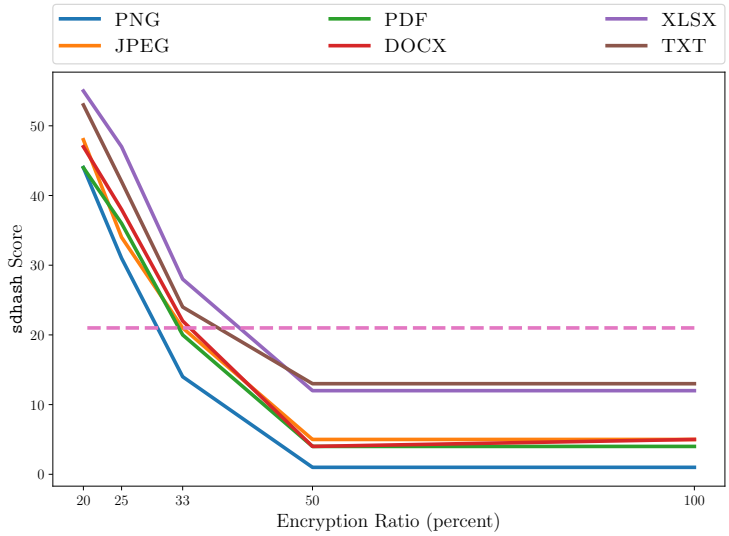

Fig. 2. Average scores of `sdhash` comparison of partially encrypted file types. Scores above 21 (denoted by the dashed line) is considered as a strong indication similarity between compared file contents.

using Eq. (2)) remains constant under any permutation as the N_i values are not altered but rearranged. As a result, the permuted data does not fit the random distribution and (**BA**-iv) does not trigger the alarm.

5 Implementation

We have developed two prototypes in order to demonstrate the feasibility of the methods described in §4. Both programs are implemented in C# language targeting version 3.5 of .NET Framework. In addition, we ported the second prototype to Python 3 (see §6).

The prototype which aims to bypass key-oriented defenses first enumerates the target files in the victim's computer. It uses an obfuscated SHA-256 function to compute hashes and the F-PRNG is initialized with 50 files' digests. This is the maximum capacity of the F-PRNG's pool which is implemented as a byte array. Our novel ransomware uses RSA algorithm for public key encryption. Once the F-PRNG is ready, two 1024 bit primes are generated, an RSA key pair is computed, and the private key is encrypted with the embedded master public key. Primality tests are performed using Miller-Rabin algorithm with the iteration count set to 3 as indicated in [19]. F-PRNG is also utilized to generate the padding values used for randomization of ciphertext.

The second prototype targets behavioral based approaches which monitors file system activities. It has two working modes: *partial* and *full* encryption. The

former targets CryptoDrop and performs partial encryption and the latter fully obfuscates files. In our design, we set block size to $n = 64$, *i.e.,* read 64 bytes, permute this block and overwrite the original data. Fisher-Yates [9] algorithm is utilized to permute the blocks. We remark that, while executing Fisher-Yates algorithm, the required randomness is obtained from the CSPRNG APIs as behavioral analysis based systems do not control these.

Both of the prototypes contain only encryption routines, file I/O functions, and codes responsible for the key management tasks. As our main purpose is to show potential attacks and not to develop a fully functional ransomware, we deliberately omitted implementing all non-cryptographic functions, such as spreading over the network and deleting the Volume Shadow Copy Service (VSS) backups. Furthermore, our prototypes save a copy of encryption key in the same directory for each encrypted file to prevent accidental damages.

6 Experimental Results

In order to verify the feasibility of the methods described in §4, we tested our prototypes against ransomware defense systems in Table 1 that provides a implementation. In this regard, we conducted a series of experiments on PayBreak, UShallNotPass, DaD and CryptoDrop.

The test environment is prepared as follows. We created a virtual machine (VM) in VirtualBox[5] and performed a clean install of 32 bit version of Windows 7 OS. Next, we created 5 directories on user desktop and randomly placed decoy files therein. The decoy set contained 10 files with each of the extensions `.docx`, `.jpg`, `.pdf`, `.png`, `.txt` and `.xlsx`, making 60 in total. Before our experiments, we confirmed that the decoy files could be opened by the associated applications and were free of any corruption. Finally, we deactivated User Access Control (UAC) and Windows Defender to prevent interference, and took a snapshot of the test system.

We started experiments by testing the first prototype against UShallNotPass. After running the executable of our first prototype, we observed that all decoy files were encrypted while the UShallNotPass was active. We rollback to the snapshot and started testing the next system, PayBreak[6]. Our prototype run and the files were encrypted, however, the log file of PayBreak did not contain any cryptographic material. As a result, we observed that our first prototype bypassed the software implementations of two key-oriented defense systems.

We continued our experiments with the behavioral analysis systems. We first tested the 32-bit version of DaD[7] against our second prototype. We activated DaD, executed the prototype and observed that all the decoy files were corrupted. Therefore, we conclude that our prototype could evade DaD.

[5] VirtualBox, https://www.virtualbox.org/
[6] Compiled from source available at: https://github.com/BUseclab/paybreak.
[7] Downloaded from http://people.rennes.inria.fr/Aurelien.Palisse/DaD.html.

Finally, we evaluated our prototype against CryptoDrop[8] as follows. Although we did not have an open source implementation of CryptoDrop, the mechanisms that [25] uses, *i.e.,* `file` and `sdhash` tools are publicly available and installable on a Linux system. Moreover entropy changes can also be monitored using `ent`[9] tool. Therefore, we re-implemented our prototype in Python 3 and run in partial encryption mode on a Linux system. We observed that `file` command reported that the original and encrypted files are of exactly same type. Moreover, all `sdhash` comparison scores were above 21 using %30 encryption. Finally, `ent` tool measured the partially-encrypted files have the same entropy with the original ones. Based on these results, we conclude that our prototype can bypass CryptoDrop.

We remark that partial encryption causes damage sufficient to make the files unusable. In our experiments we observed that images could not be rendered and documents could not be read even with 20% encrypted files. Only exception is the TXT files that we could read the non-encrypted contents.

7 Conclusion, Discussion, and Future Work

The purpose of this work is to warn the scientific community of forthcoming ransomware threats. By talking about how seven cutting-edge anti-ransomware solutions —at the time of this writing, implementing strategies of access control over random number generators, key escrow, and behavioral analysis are the most advanced strategies known against active ransomware samples— could be overthrown by smarter and more sophisticated malware, we hoped to have revealed what strategies those malware could trying to implement, so indicating where anti-ransomware engineers have to focus their efforts. Since it is believed that the ransomware threat will increase not in number of attacks but in sophistication, to keep anti-ransomware ideas ahead of time may be a game-changing factor.

That said, malware mitigation is an arms race and we expect new generations of ransomware coming soon with renovated energy and virulence, adapting their attack strategies to challenge current defenses. New variants of ransomware have been observed constantly during the last years. Those called *scareware* prefer to exploit people's psychology, threatening them into pay the ransom without, however, doing any serious encryption: despite deceitful they are technically benign applications. Others, however, will be variants of real cryptographic ransomware and able to overcome control and to encrypt a victim's files using strong encryption. A recent white paper by Symantec [29] reports that ransomware is becoming instrument for specialists and targeted attack groups, a weapon

[8] This paper analyzes the academic paper version of CryptoDrop [25]. The software available at https://www.cryptodrop.org/ is a proprietary & commercial product, and its source code is not available. It may include undocumented measures other than the ones in the academic paper, therefore, we could not inspect the code nor analyze the actual implementation in this study.

[9] ENT: A Pseudorandom Number Sequence Test Program, http://www.fourmilab.ch/random/

not only to extort money but to cover up other attacks and, when using strong encryption, used in fact as a disk wiper. It is to this latter category that our research is dedicated. As security professionals we feel compelled to be prepared to face forthcoming threats thus to identify and anticipate potentially dangerous ransomware variants, and warn the scientific community about them.

We are aware that the research we have ourselves embarked may give ideas to criminals. But there is no reason to believe that criminals will not have those ideas by themselves. In the history of malware (see *e.g.,* [11]) criminals have always tried to be one step ahead; besides, our research has nothing fancy and it does not contain such an inventive step that cannot be reproduced by others. It more humbly roots into how cryptography works. However, even with this premise, we questioned ourselves about how to do this research ethically.

7.1 Ethical Code of Conduct

As we anticipated in the introduction, working with malware raises ethical questions [28], although we have not involved people in our research, nor we have collected personal or sensitive data or attacked real operating systems, nor were we involved in any conversations with criminal associations or victims, actions which would have required us following specific guidelines as discussed in [5].

Despite having conducted our research in isolation, we agree with Rogaway's "The Moral Character of Cryptographic Work" [22] when he suggest to "be introspective about why you are working on the problems you are". We hope to have motivated sufficiently why we started this research pathway in the first place. At the same time we informed ourselves about the University of Luxembourg Policy on Ethics in Research[10]; it suggests that researching on protection against computer viruses is at risk of dual use. The guidelines recommend researchers to "report their findings responsibly", but there is no indication that may suggest what is a responsible behavior. As well there are no guidelines in the ACM Code of Ethics and Professional Conduct[11], another manifesto we looked into. It suggests principles, like "Avoid harm" and "Ensure that the public good is the central concern during all professional computing work" but how to comply with those principles is not told. The EU "Regulation No 428/2009" considers software as a dual use item, so we are certain that there are ethical consideration to address. Most of the literature on dual-use refers to life science and cannot be migrated to computer science but the EU's "Ethics for researchers" [6] suggests something general that can be useful in our case: "special measures need to be taken to ensure that the potential for misuse is adequately addressed and managed". Thus we decided to set up our own ethical practise which consist in embrace two important measures: (i) *Responsible Disclosure*: before submitting camera ready version, we informed all parties affected by the vulnerabilities that we think we have disclosed in this paper, giving them all details about the flaws

[10] For more information, please visit https://wwwen.uni.lu/research/chercheurs_recherche/standards_policies

[11] Available at https://www.acm.org/code-of-ethics

and the potential attacks. We hope in this way to warn awareness in the scientific community, and in particular in the researchers that engineered the defences whole limitations we have discussed; (ii) *Safe Handling of Hazardous Code*: we determined ourselves not to share any portion of the source code with the public, not to send it unsecured in using insecure channels (*e.g.,* emails) and to keep it stored in an encrypted disk. At the same time all experiments have been done with a machine whose access is strictly limited to the researchers involved.

7.2 Limitations and Future Work

Current BA systems use statistical tests to detect encryption. To evade this protection, we had to use pure permutation to obfuscate files and this is definitely not as secure as standard ciphers, *e.g.,* AES algorithm. If the permutation can be discovered practically, the ransomware cannot force the victims to pay. However, the question is still open: does it provide the minimal security level in the context of ransomware, *i.e.,* decrypting might be possible but paying the ransom is more economic than decrypting? Due to space restrictions, we leave this task for a future work.

Pure permutation technique is successful against (**BA**-i) and (**BA**-iv). Moreover, it can be adopted to evade (**BA**-ii) and (**BA**-iii). Other systems, [14,2,15] watch additional indicators to detect ransomware activity. We leave the task of evaluating the feasibility of evading these indicators to a future research.

To the best of our belief, this work is the first one that proposes to gather entropy from file contents in order to generate prime numbers; but we restricted ourselves to achieve this aim by using merely a hash function. We remark that the security of RSA key pair generation method should be carefully studied.

References

1. Barkly: 2017 Ransomware Report. Tech. rep., Barkly (2017)
2. Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., Maggi, F.: ShieldFS: A Self-healing, Ransomware-aware Filesystem. In: Proc. of the 32Nd Annual Conf. on Computer Security Applications. pp. 336–347. ACM, New York, NY, USA (2016)
3. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag, Berlin, Heidelberg (2002)
4. Darwin, I.: Fine Free File Command, http://www.darwinsys.com/file/
5. Deibert, R., Crete-Nishihata, M.: Blurred Boundaries: Probing the Ethics of Cyberspace Research. Review of Policy Research **28**(5), 531–537 (2011)
6. Directorate-General for Research and Innovation: Ethics for Researchers Facilitating Research Excellence in FP7. Tech. rep., European Commission (July 2013)
7. Douceur, J.R., Adya, A., Bolosky, W.J., Simon, D., Theimer, M.: Reclaiming Space from Duplicate Files in a Serverless Distributed File System. In: Proc. of the 22 Nd Int. Conf. on Distributed Computing Systems. pp. 617–624. IEEE, Washington, DC, USA (2002)
8. Eastlake 3rd, D.: Publicly Verifiable Nominations Committee (NomCom) Random Selection. RFC 3797 (June 2004), https://tools.ietf.org/pdf/rfc3797.pdf

9. Fisher, R.A., Yates, F.: Statistical Tables for Biological, Agricultural and Medical Research. Oliver and Boyd (1938)
10. Genç, Z.A., Lenzini, G., Ryan, P.Y.: No Random, No Ransom: A Key to Stop Cryptographic Ransomware. In: Proc. of the 2018 Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham (2018)
11. Herrera-Flanigan, J.R., Ghosh, S.: Criminal regulations. In: Cybercrimes: A Multidisciplinary Analysis, pp. 265–308. Springer (2011)
12. Hirschberg, B., Kravchik, M., Haenel, A., Solow, H.: Ransomware Key Extractor and Recovery System (April 2016), https://patentscope.wipo.int/search/en/detail.jsf?docId=US215058675
13. Kaspersky: KSN Report – Ransomware in 2014-2016. Tech. rep., Kaspersky (2016)
14. Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., Kirda, E.: UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In: 25th USENIX Security Symposium. pp. 757–772. USENIX Association, Austin, TX (2016)
15. Kharraz, A., Kirda, E.: Redemption: Real-Time Protection Against Ransomware at End-Hosts. In: Research in Attacks, Intrusions, and Defenses. pp. 98–119 (2017)
16. Kim, H., Yoo, D., Kang, J.S., Yeom, Y.: Dynamic ransomware protection using deterministic random bit generator. In: 2017 IEEE Conference on Application, Information and Network Security (AINS). pp. 64–68 (Nov 2017)
17. Kolodenker, E., Koch, W., Stringhini, G., Egele, M.: PayBreak: Defense Against Cryptographic Ransomware. In: Proc. of the 2017 ACM on Asia Conf. on Computer and Communications Security. pp. 599–611. ACM, New York, USA (2017)
18. Lee, K., Oh, I., Yim, K.: Ransomware-Prevention Technique Using Key Backup. In: Jung, J.J., Kim, P. (eds.) Big Data Technologies and Applications. pp. 105–114. Springer (2017)
19. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1st edn. (1996)
20. Palisse, A., Durand, A., Le Bouder, H., Le Guernic, C., Lanet, J.L.: Data Aware Defense (DaD): Towards a Generic and Practical Ransomware Countermeasure. In: Secure IT Systems. pp. 192–208. Springer, Cham (2017)
21. Palisse, A., Le Bouder, H., Lanet, J.L., Le Guernic, C., Legay, A.: Ransomware and the Legacy Crypto API. In: 11th International Conference on Risks and Security of Internet and Systems - CRiSIS. pp. 11–28. Springer (Sep 2016)
22. Rogaway, P.: The Moral Character of Cryptographic Work. Cryptology ePrint Archive, Report 2015/1162 (2015), https://eprint.iacr.org/2015/1162
23. Roussev, V.: Data Fingerprinting with Similarity Digests. In: Chow, K.P., Shenoi, S. (eds.) Advances in Digital Forensics VI. pp. 207–226. Springer (2010)
24. Roussev, V., Quates, C.: The sdhash tutorial (2013), http://roussev.net/sdhash/tutorial/03-quick.html
25. Scaife, N., Carter, H., Traynor, P., Butler, K.R.B.: CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). pp. 303–312 (June 2016)
26. Stark, P.B.: Pseudo-Random Number Generator using SHA-256, https://www.stat.berkeley.edu/~stark/Java/Html/sha256Rand.htm
27. Steve Morgan: 2017 Cybercrimes Report. Tech. rep., Cybersecurity Ventures (2017)
28. Sullins, J.P.: A Case Study in Malware Research Ethics Education: When Teaching Bad is Good. In: Proc. of IEEE Security & Privacy, 17-18 May 2014, San Jose, CA, USA. IEEE computer society (2014)
29. Symantec Corporation: Internet Security Threat Report. Tech. rep. (April 2018)
30. Touchette, F.: The Evolution of Malware. Network Security **2016**(1), 11–14 (2016)