Security Analysis of Key Acquiring Strategies Used by Cryptographic Ransomware

Ziya Alper Genç
University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
ziya.genc@uni.lu

Gabriele Lenzini
University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
gabriele.lenzini@uni.lu

Peter Y.A. Ryan
University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
peter.ryan@uni.lu

ABSTRACT

To achieve its goals, ransomware needs to employ strong encryption, which in turn requires access to high-grade encryption keys. Over the evolution of ransomware, various techniques have been observed to accomplish the latter. Understanding the advantages and disadvantages of each method is essential to develop robust defense strategies. In this paper we explain the techniques used by ransomware to derive encryption keys and analyze the security of each approach. We argue that recovery of data might be possible if the ransomware cannot access high entropy randomness sources. As an evidence to support our theoretical results, we provide a decryptor program for a previously undefeated ransomware.

CCS CONCEPTS

• **Security and privacy → Key management**; **Malware and its mitigation**;

KEYWORDS

ransomware, key acquiring, security analysis, malware

ACM Reference Format:
Ziya Alper Genç, Gabriele Lenzini, and Peter Y.A. Ryan. 2018. Security Analysis of Key Acquiring Strategies Used by Cryptographic Ransomware. In *Central European Cybersecurity Conference 2018 (CECC 2018), November 15–16, 2018, Ljubljana, Slovenia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3277570.3277577

1 INTRODUCTION

"Many of your documents are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service."

This is the opening message of WannaCry, a cryptographic ransomware that become known to the general public in May 2017 after having encrypted documents on thousands of computers around the world. The message is crafted to intimidate. Since the ultimate goal of a ransomware is to collect money, the message should sound both convincing and threatening to a victim, letting him/her ponder the pros to pay up a well calculated, but not huge, sum of cryptocurrency against the cons to loose irremediably access to valuable documents. Whatever the decision the message is clear: there is no other way out. One question emerges naturally: is it true that a victim is left only with this "pay or perish" dichotomic choice?

Let us assume that the ransomware has really encrypted the files, that is, let us consider out of scope those malware called scareware and that just pretend to have encrypted the victim's documents. As cryptographers we know that decrypting without the key can be indeed an intractable problem if the key used for the encryption is unpredictable, that is, both strong and secret. But we also know that obtaining good keys and keep them safe are hard tasks; even good cryptographic applications can fail to do it well [1]. The situation should not be better for ransomware and we expect some implementation be vulnerable exactly in the way they try to cope with those hard tasks.

In this paper we study the various strategies that current ransomware follows to acquire cryptographically strong encryption keys and to keep them safe from anti-malware, and we comment on how good or bad those strategies are. Wherever they even partially fail, we discuss whether and how the encrypted files can be recovered. In this way, we reveal when and for which ransomware the threatening statement "nobody can recover your file" really holds true and when instead it sounds more threatening than it really is.

The rest of the paper is structured as follows. §2 discusses how ransomware works in general and classify the methods of acquiring encryption keys. Previous works related to our study is summarized in §3. In §4, key-oriented protection systems are reviewed. The main result of our analysis is presented in §5. In §6, we demonstrate evidence from real-world examples to support our result and provide a decryptor for a ransomware sample which was not available previously. §7 discusses the future works and concludes the paper.

2 BACKGROUND

2.1 Generic Ransomware Functionality

We focus only on the cryptographic aspects: other malicious behaviours such as spreading over the network and altering OS configuration is out of the scope of this paper. That said, after infection, a ransomware commences preparation phase, in which it enumerate the files it intends to target target and start acquiring or building the encryption keys. Next, the ransomware proceeds to encrypt files. Once the encryption finishes, it notifies the victim and delivers the ransom demand. This operation flow is depicted in Fig. 1.

CECC 2018, November 15–16, 2018, Ljubljana, Slovenia

ACM ISBN 978-1-4503-6515-4/18/11…$15.00
https://doi.org/10.1145/3277570.3277577

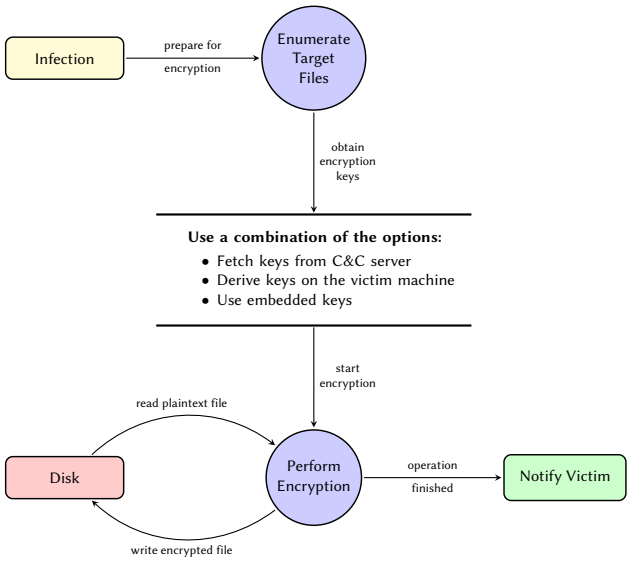

Figure 1: Operation flow diagram of a ransomware from the cryptographic point of view.

To encrypt victim's files, ransomware usually employs a hybrid cryptosystem, that is, an encryption scheme consist of a combination both symmetric and asymmetric algorithms. In order to understand why ransomware authors need to use hybrid cryptosystems, one needs to observe the following facts:

- Encrypting files with solely asymmetric algorithms is a resource intensive task. This might introduce the risk of being detected as high CPU usage for a long time could trigger anomaly detection systems. Therefore, ransomware must utilize a symmetric algorithm to encrypt files.
- Ransomware needs to use a unique key for each target to prevent victims helping each other [7]. In a mass infection, managing the keys with solely symmetric primitives would not be scalable. Therefore, use of an asymmetric algorithm is required while maintaining the key management of a ransomware campaign.

2.2 Acquiring Encryption Keys

To achieve long term success, ransomware needs to exercise strong encryption, which requires to use good encryption keys. During the evolution of ransomware, various techniques have been observed to accomplish this task.

One strategy of acquiring the public keys is to fetch them from command-and-control (C&C) servers as CryptoLocker does [18]. Ransomware can also obtain file encryption keys from these C&C servers, however, usually these keys are derived from the outputs of Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) functions, *e.g.,* WannaCry [19]. Another option is to utilize a non-cryptographic Pseudo-Random Number Generator (PRNG) to seed key derivation algorithms. Programming languages in fact provide PRNG functions for developers to utilize in various domains. For example, a ransomware can call `rand` function in C standard library or use `System.Random` class in C# language to obtain pseudo-random numbers. This is more frequently observed in samples developed in C# language such as NegozI [20] and Rush/Sanction families [13]. Lastly, ransomware authors might embed the encryption keys into the malware body before spreading it, *e.g.,* Cryzip [15]. These keys can be used directly to encrypt files and thus removing the need for calling any PRNG.

Using the observations above, we identify the following methods that are used by the current generation of cryptographic ransomware to obtain encryption keys:

M1 Derive keys from CSPRNG outputs
M2 Fetch encryption keys from remote servers
M3 Utilize non-cryptographic PRNG to generate secrets
M4 Use secrets embedded into binary executable

3 RELATED WORK

Acquiring encryption key is a *sine qua non* task for a cryptographic ransomware, so discussed in detail in the technical analysis sections of previous research. In one of the earliest works on ransomware, Gazet [7] reverse engineered samples from a few families, including Gpcode, and examined the key derivation methods. Later, Kharraz *et al.* [10] performed an analysis of 1359 samples from 15 families which reports, among other findings about ransomware, different techniques used for key generation. In addition, Cabaj and Mazurczyk proposed a dynamic blacklisting system which tries to prevent delivery of public keys by disrupting the communication with C&C servers to mitigate ransomware [2]. In another study, Craciun *et al.* [3] analyzed how recent ransomware variants evolved, taking into account their key management schemes. That said, these works treat the functional aspects of the key acquiring task and focus on the evolution of the ransomware.

Apart from the scientific reports, several protection systems are proposed in the literature, approaching to the ransomware problem by focusing on the encryption keys. We describe these systems and discuss their pros and cons in §4. In addition, Cabaj and Mazurczyk analyzed the key acquiring scheme used by CryptoWall and proposed a dynamic blacklisting system which tries to prevent delivery of public keys by disrupting the communication with C&C servers [2].

Possibly, the closest work to our research is the taxonomy of key management models of ransomware, presented by Bajpai *et al.* [6]. The authors introduce a classification for ransomware based on the employed key management architecture and evaluate the potency of each category. The present work differentiates from the previous one by focusing on the *security* of key acquiring methods, and offers a recovery tool by exploiting a weakness, thereby providing both theoretical and practical contributions.

4 KEY–ORIENTED PROTECTION METHODS

Among the key acquiring methods, M1 deserves particular attraction as it is the *de facto* technique for deriving encryption keys and other cryptographic materials. Consequently, there have been several defense techniques to ransomware from including this powerful weapon to its arsenal.

Escrowing Keys. Key escrow is the practice of storing keys and cryptographic materials with the purpose of using at a later time.

In the context of ransomware, key escrow implies to capture the secret keys and other parameters while the ransomware is executing an encryption algorithm to recover the files after the attack. PayBreak [12] implements this approach by hooking the cryptographic functions provided by operating system (OS) and third-party libraries. The hook functions obtain the parameters of said Application Programming Interfaces (APIs) and store in a secure key vault. These values are used to recover the encrypted files after a ransomware attack. On the other hand, as the authors state, ransomware can evade PayBreak by statically linking third-party libraries and obfuscating the executable code. Even in this case, PayBreak would log the CSPRNG outputs.

Replacing CSPRNG. Replacing CSPRNG functions with a backdoored PRNG, enables defences to reproduce the outputs obtained by applications that called these functions. Kim *et al.* proposed such a strategy [11]. They replace the CSPRNG of the host system with a user-defined number generator. Called *deterministic random bit generator* (DRBG), this PRNG contains a trapdoor which allows the user to retrieve the outputs of DRBG at a given time. The trapdoor is a secret value, preferably stored in an external device and assumed to be accessible only by the user. If a ransomware attack occurs, the trapdoor value is used to reproduce the outputs of DRBG, thus derive the encryption keys used by ransomware and recover the files. This defence can bypassed by finding other ways than calling CSPRNG to derive keys.

Controlling CSPRNG. Obtaining true randomness is a difficult task, so modern operating systems expose dedicated APIs to serve for this purpose. These built-in functions are called CSPRNG and are of limited numbers. Combining this fact with the techniques that OSs provide to hook APIs, it becomes feasible to control CSPRNGs. This is what a new anti-ransomware, UShallNotPass [8], does to prevent unauthorized applications from using CSPRNG functions. It intercepts calls made to CSPRNG of host system and allows access only to a whitelisted processes, terminating all the others. Finding an alternative randomness source may allow to evade this protection.

5 MAIN RESULT

We now analyze the security of the four methods presented in §2. In our discussions, we will refer to the following categorization of the key-related weaknesses:

- **W1** Reproducibility of the keys: files might be decrypted by reproducing the encryption keys used by ransomware.
- **W2** Failure to derive keys: ransomware might fail to derive keys due to an access control over CSPRNG of the host system.
- **W3** Interception of transportation: a defense mechanism on target system might be able to prevent the ransomware from acquiring keys over a network location.
- **W4** Guessable encryption keys: finding the encryption keys might be feasible by exploiting a cryptographic weakness in the scheme used by ransomware.
- **W5** Extractable secret values: reverse engineering the executable will enable extracting the secrets to restore the files.

5.1 Deriving Keys from CSPRNG Outputs

As we described in §4, several different approaches exist in the literature to protect CSPRNG It can be seen that if one of the systems in §4 is active on the target machine at the time of attack, W1 and W2 can be exploited against a ransomware that belongs to the category M1, and victim's data can be restored.

5.2 Using Embedded Secrets

While embedding the keys and other cryptographic materials into the executable (M4) seems to be the easiest method from the development point of view, extracting these secrets would enable reversing the actions ransomware performed, *e.g.,* building a decryptor which would defeat that ransomware campaign. In fact, early ransomware (*e.g.,* Cryzip [15]) used this approach which brought about NoMoreRansom Project, providing decryptors for several ransomware families. Furthermore, the practice of using embedded keys can still be seen in recent samples[1] and easily thwarted. Ransomware might try to destroy itself to keep the confidentiality of the keys, however, the binary executable might be re-obtained from the intrusion point. Therefore, we conclude that if the ransomware authors build their scheme on a secret in the malware body, failure of their campaign is inevitable.

5.3 Utilizing a Non-cryptographic PRNG

Another technique that ransomware might employ is to derive keys from a non-cryptographic PRNG (M3). For example, the key generation algorithm of NegozI [20] and Rush/Sanction [13] families is demonstrated in Listing 1.

```
public static string GetPass(int x)
{
   string str = "";
   Random random = new Random();

   while (str.Length < x)
   {
      char c = (char) random.Next(33, 125);
      if (char.IsLetterOrDigit(c))
      str += c;
   }

   return str;
}

var password = GetPass(new Random().Next(30, 50));
```

Listing 1: Password generation method used by NegozI and Rush/Sanction ransomware families.

In this strategy, ransomware families restrict themselves to build a password using only digits and characters. The reason is that victims may not be able to type non-printable characters (like control sequences) when they are asked to enter the password for decryption. This would cause the password to be built up from a low entropy set. In order to fix this shortcoming, ransomware authors usually select a password in the range of 30 to 50 characters. The total entropy (in bits), e, of a password can be computed as:

$$e = \log_2 C^{\ell} \tag{1}$$

[1] https://malwareless.com/remove-rarucrypt-ransomware-virus-removal-guide

where C is the size of the character set and ℓ is the length of the password. Using Eq. (1), the entropy of the passwords returned from `GetPass` can be estimated as follows:

$$\log_2 62^{30} \leq e \leq \log_2 62^{50}$$

As a result, the entropy of the passwords would lie in 178.62 bits and 297.71 bits, and cracking them using brute-force would not be feasible. However, this approach has a critical flaw which significantly reduces the complexity. Non-cryptographic PRNGs require a seed value, which, if given again , causes the PRNG to produce the same output. If the seed is a static secret, then it can be extracted as described in the previous subsection and the same output stream can be generated. Alternatively, the PRNG could be fed by the system time which could be accurately estimated by forensics tools allowing to generate the same outputs. Trying to obtain more entropy from other sources (monitoring mouse activity or keyboard inputs) would trigger the anti-virus solutions, *i.e.,* the malware would be marked as a keylogger and neutralized. We elaborate on this weakness in §6.

5.4 Fetching Keys from C&C Center

Different from previous approaches, ransomware might choose to download encryption keys from C&C servers [14]. Especially, samples which utilize asymmetric algorithms prefer this method as it allows the private keys to remain at attacker's side which provides a better security for ransomware. However, this requires the C&C server to be reachable from the victim's machine at the attack time. Based on this fact, Cabaj and Mazurczyk [2] proposed a real-time network monitoring system which dynamically blacklist C&C server's IPs to block communication between the victim and the attacker (W3).

Observation on Usability: Choosing M2 as a primary method requires a fail-safe mechanism for acquiring encryption keys. For this reason, M1, M3, or M4 are employed as a backup strategy. Indeed, Spora and Sage families, in addition to new versions of Locky and Cerber, contain an offline encryption mode by default [17].

5.5 Discussion

In the light of our analysis, each method presented in §2 has a weakness that can be exploited to combat ransomware. Table 1 shows each methods against the corresponding weaknesses.

Table 1: Weak points of the key acquiring methods.

Method	Weakness				
	W1	W2	W3	W4	W5
M1	•	•			
M2			•		
M3	•			•	
M4					•

We argue that by placing the correct countermeasures on the host system before infection, recovery of the files might be feasible. In particular, if CSPRNG of the host system is protected, failure of the ransomware attack is inevitable. In other words, if the randomness comes from a weak source, *i.e.,* not cryptographically secure keys are likely to be predictable and can be found by brute-force. Thesis 1 systematically states this argument:

Thesis 1. *Let R be a ransomware sample which employs a hybrid scheme for encrypting victim's files. If R cannot access the CSPRNG of the victim's computer, then R cannot obtain good encryption keys. Recovery without paying the ransom may be possible.*

Argument. Since R uses a hybrid cryptosystem, it employs a symmetric encryption algorithm, say E, to encrypt victim's files. To use in this algorithm, ransomware also needs at least one secret key K. This observation sits in the central of our logic.

We accept the premise in the thesis and assume that R cannot use the method M1 to obtain K. In addition, M2 can be neutralized as it has the weakness W3. Moreover, M3 and M4 cannot be used to obtain secure keys due to the weaknesses W4 and W5, respectively. As a result, R cannot acquire secure encryption keys.

Now we argue that the recovery of the files without paying the ransom might be possible. In order to generate K, R might use M3 or M4, as M1 is not available and M2 is blocked. If R uses M3, *i.e.,* utilizes a non-cryptographic PRNG to seed its key derivation function, the seed would be guessable which allows the files to be restored. R can also try to obtain the seed values from the environment and mix them, but this would trigger the anti-virus systems. Therefore, the PRNG will be used with a predictable seed and its outputs can be reproduced. Finally, R can employ M4, *i.e.,* use the key K and other cryptographic materials embedded into the malware body. In this scenario K can be extracted by reverse engineering methods. Using packers or other binary protection techniques can only make the process take longer.

As a result, R cannot obtain strong encryption keys and it is likely that original files can be restored without paying the ransom. ■

6 REAL WORLD EXAMPLES

We now provide empirical results obtained from real-world ransomware samples to support at least with some evidence the argument we uphold for Thesis 1.

We obtained malware samples by performing a search on Hybrid Analysis[2] with the tags *ransomware*, *cryptovirus* and file-type substring *PE32 executable (console)*. Next, we prepared a test environment to confirm the collected samples are active ransomware, following the previous work [8].

For automated testing, we utilized the Cuckoo Sandbox[3] open source automated malware analysis system. We had UShallNotPass running in passive mode to log CSPRNG usage during the tests. Active ransomware samples are identified by checking the hashes of decoy files after each test. Among the them, we picked a sample which does not call CSPRNG APIs and does not have a decryptor available. To identify the sample, AVClass tool [16] is employed, which determines the family name by performing plurality vote on the labels assigned by AV engines. We obtained these labels from VirusTotal[4], and found that the sample[5] is a Crypren variant.

[2]Hybrid Analysis, https://www.hybrid-analysis.com
[3]Cuckoo Sandbox, https://cuckoosandbox.org
[4]VirusTotal Intelligence, https://www.virustotal.com
[5]SHA-1 hash of the sample is `18e49f01a7493ea56f520ea5cbaf43ca2daca71c`.

Using CFF Explorer[6], we detected that the ransomware is written in C# language. Finally, to analyze the sample, we decompiled the executable binary into source code using dotPeek[7] tool.

6.1 Extracting Secret Phrase from Binary

The Crypren sample analyzed in this paper performs various malicious operations before encrypting the victim's files, including deleting Volume Shadow Copy Service (VSS) backups quietly, impersonating another user, and creating a new Windows account. The `Main` function of the sample which is depicted in Listing 2. We analyze the code sections, which perform key generation and encryption of files.

```
public static void Main (string[] args)
{
   ...
   string password = Program.GetEncKey();
   List<string> files = new List<string>();

   foreach (string str in files)
   {
      string encFile = file + ".enc";
      new Thread((ThreadStart) (() => Program.EncryptFile(file,
      ↪  encFile, password))).Start();
       ...
```

Listing 2: Code snippet from the Main function.

The sample invokes `GetEncKey` to get the password which will be used to encrypt victim's files. Once the key is obtained, the sample enumerates all the drives on the victim system and looks for the files with targeted extensions. Next, it creates a thread that executes `EncryptFile` for each file. It is remarkable that all files are encrypted using a single key. We now move to Listing 3 which gives the implementation of the function `GetEncKey` to see how this Crypren variant acquires the encryption keys.

```
private static string GetEncKey()
{
   try
   {
      using (WebClient webClient = new WebClient())
         return webClient.DownloadString(
         ↪  "http://ohad.000webhostapp.com/cnc.php?txt=saveme")
         ↪  .Trim();
   }
   catch
   {
      return "myke123!";
   }
}
```

Listing 3: Implementation of the `GetEncKey` function.

The sample does not call `CryptGenRandom` API, rather follows a strategy that employs a combination of the methods M2 (though not for public key, but for a secret phrase) and M4 to acquire the encryption key. First, the sample attempts to connect the hard-coded address of its C&C server and download a string data from there, *i.e.,* M2 is the preferred key-acquiring method, and the server transmits a string data. If this fails, Crypren uses the secret phrase embedded in its code (M4), as a fail-safe mechanism. Once the password is acquired, the ransomware starts to encrypt victim's files by executing `EncryptFile` which is given in Listing 4.

```
private static void EncryptFile(string inputFile, string outputFile,
↪  string password)
{
   try
   {
      byte[] bytes = new UnicodeEncoding().GetBytes(password);
      ...
      CryptoStream cryptoStream = new CryptoStream((Stream)
      ↪  fileStream1, rijndaelManaged.CreateEncryptor(bytes,
      ↪  bytes), CryptoStreamMode.Write);
      ...
}
```

Listing 4: Code snippet from EncryptFile function.

The encryption key is computed by encoding the password into a byte array under UTF-16 format, *i.e.,* no cryptographic function is utilized in key derivation. The sample employs `RijndaelManaged` class which provides encryption routines of the AES [4]. No cipher mode is set explicitly, so the default mode, Cipher Block Chaining (CBC) is used. Initialization vector (IV) equals to the value of encryption key, again omitting to call `CryptGenRandom` API.

Decryptor. Using these results, we have implemented a decryptor for the Crypren sample analyzed in this paper. Our prototype picks an encrypted PDF file and performs *brute-force attack* to find the correct password. At first step, our prototype tries to decrypt the first block of the ciphertext with the embedded secret `myke123!`. If the result looks like the valid PDF header, *i.e.,* starts with `0x25504446`, then the password is found and operation continues with the next file. Otherwise, the decryptor starts an exhaustive scan to find the correct password. The domain of the secret phrase is inferred from the embedded password in which we distinguished lower case letters, numbers and special characters[8]. Consequently, using Eq. (1), we estimate the entropy of the password space as $\log_2 46^8 = 44.19$ bits. On Intel i7 CPU clocked at 3.6 GHz, and leveraging AES-NI instructions, we achieved to try 117.03M pass/s which enables to find the password in 66.89 hours in the worst case. Our prototype is an open source software[9] developed in C# language and will be shared with No More Ransom and ID Ransomware platforms.

6.2 Guessing Keys Derived from Weak RNG

In §5, we presented an example of non-cryptographic PRNG usage seen in NegozI and Rush/Sanction families. Using `Random` class to generate passwords is not uncommon for ransomware developed in C# language. However, according to the analyses [13, 20], these ransomware variants dispose the keys without saving them for recovery. After the files are encrypted, victim's only option is to search for a flaw in the ransomware which may allow to recover the files.

For each file, NegozI and Rush/Sanction generate a unique password using `GetPass` given in Listing 1. This secret phrase is used

[6] CFF Explorer, http://www.ntcore.com/exsuite.php
[7] dotPeek, https://www.jetbrains.com/decompiler

[8] We consider the following special characters `!@#$&*-+=?`.
[9] Available under GPLv3 at https://github.com/ziyagenc/crypren-decryptor.

to derive the encryption keys and IVs using `Rfc2898DeriveBytes` class which implements PBKDF2 [9] functionality. Salt value used in PBKDF2 is hardcoded in the ransomware body and iteration count is set to 1000. Each file is encrypted with a different key and IV utilizing `RijndaelManaged` class.

The output of `GetPass` function, even generated using a non-cryptographic PRNG, is sufficiently disincentive for brute-forcing even when the minimum length of the password is set to 30. However, the flaw in these families enables the recovery process to complete significantly faster, by narrowing down the password space so making the brute-forcing practical, as follows. Given the same seed, `Random.Next` method in .NET framework produces the same output. Moreover, if `Random` class is instantiated parameterless, the constructor uses `Environment.TickCount` as a seed value[10], *i.e.,* system uptime. Windows OS periodically logs system uptime in event logs, so can be obtained after the attack. Furthermore, the infection time can be retrieved from file system by looking for file write time. Combining these two clues eliminates considerable portion of the uncertainty of the seed value. Using this advantage, researchers developed a decryptor[11] for NegozI ransomware which can recover files in hours.

7 CONCLUSIONS AND FUTURE WORK

The ransomware business model depends on the encrypted data not being recoverable without paying the ransom. This relies on two ingredients: robust cryptographic algorithms and strong encryption keys. The first is easy: ransomware authors can use cryptographic functions provided by the OS, or publicly available third party libraries, *e.g.,* OpenSSL. Fortunately (for the victims), obtaining good encryption keys is a difficult task. Thus, we evaluated how current ransomware achieve this task of acquiring cryptographic keys. We showed that there is one secure way, and that is by having access to CSPRNG APIs of the host system. Blocking that access, as the UShallNotPass [8] anti-ransomware does, leaves only weaker options: ransomware applications that are forced to use these alternatives, we argue, can have their work reversed and their targeted files decrypted. In support to this argument, we analyzed a Crypren sample and developed a decryptor which was not available before. We also reported and discussed the weaknesses in the key generation methods used in NegozI and Rush/Sanction families.

There is future work to do. Cyber-criminals may develop alternative methods to generate strong encryption keys. To anticipate this evolution, we are also studying alternative ways to derive encryption keys, for instance from files as it is done in convergent encryption [5], a technique applied in cloud computing to build keys for symmetric algorithms. That said, the security of generating asymmetric keys from files is not studied yet, and we think it is an interesting research topic to investigate further.

In §2 we showed how to recover files using a method, M4, that requires manual intervention. We have to develop an automated way to accomplish this task. Lastly, due to space restrictions, we could not discuss how ransomware downloads symmetric keys from C&C servers. We intend to show that also for this option be secure, ransomware needs to call CSPRNG APIs of the host system, a strategy that UShallNotPass anti-ransomware can nullify.

[10] https://referencesource.microsoft.com/#mscorlib/system/random.cs.
[11] Available at https://github.com/zisk/evil-decrypter.

ACKNOWLEDGMENTS

This work is supported by the Fonds National de la Recherche under Grant No.: PF18/12536861/NoCry.

REFERENCES

[1] Luciano Bello. 2008. Debian Security Advisory: DSA-1571-1 openssl – predictable random number generator. (13 May 2008). Retrieved July 9, 2017 from http://www.debian.org/security/2008/dsa-1571
[2] Krzysztof Cabaj and Wojciech Mazurczyk. 2016. Using Software-Defined Networking for Ransomware Mitigation: The Case of CryptoWall. *Netwrk. Mag. of Global Internetwkg.* 30, 6 (Nov. 2016), 14–20.
[3] Vlad Constantin Craciun, Andrei Mogage, and Emil Simion. 2018. Trends in design of ransomware viruses. Cryptology ePrint Archive, Report 2018/598. (2018). https://eprint.iacr.org/2018/598.
[4] Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael.* Springer-Verlag, Berlin, Heidelberg.
[5] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. 2002. Reclaiming Space from Duplicate Files in a Serverless Distributed File System. In *Proc. of the 22 Nd Int. Conf. on Distributed Computing Systems (ICDCS '02).* IEEE Computer Society, Washington, DC, USA, 617–624.
[6] Richard Enbody, Aditya K. Sood, and Pranshu Bajpai. 2018. A key-management-based taxonomy for ransomware. In *Proc of the 2018 APWG Symp. on Electronic Crime Research, eCrime 2018*, Vol. 2018-May. IEEE Computer Society, Washington, DC, USA, 1–12.
[7] Alexandre Gazet. 2010. Comparative analysis of various ransomware virii. *Journal in Computer Virology* 6, 1 (01 Feb 2010), 77–90.
[8] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. 2018. No Random, No Ransom: A Key to Stop Cryptographic Ransomware. In *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2018).* Springer International Publishing, Cham, 234–255.
[9] Burt Kaliski. 2000. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898. (Sept. 2000). https://doi.org/10.17487/RFC2898
[10] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. 2015. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, Cham, 3–24.
[11] Haeun Kim, Dongchang Yoo, Ju-Sung Kang, and Yongjin Yeom. 2017. Dynamic ransomware protection using deterministic random bit generator. In *Conf. on Application, Information and Network Security (AINS).* IEEE, Piscataway, New Jersey, 64–68.
[12] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. 2017. PayBreak: Defense Against Cryptographic Ransomware. In *Proc. of the 2017 ACM on Asia Conf. on Computer and Communications Security (ASIA CCS '17).* ACM, New York, NY, USA, 599–611.
[13] MalwrPost. 2016. Technical Analysis of Rush/Sanction Ransomware. (6 April 2016). Retrieved June 07, 2018 from https://malwrpost.wordpress.com/2016/04/06/technical-analysis-of-rush-sanction-ransomware/
[14] Trend Micro. 2016. Network Solutions to Ransomware – Stopping and Containing Its Spread. (8 Sept. 2016). Retrieved July 9, 2018 from https://blog.trendmicro.com/trendlabs-security-intelligence/network-solutions-ransomware-stopping-containing-spread/
[15] Kevin Savage, Peter Coogan, and Hon Lau. 2015. The evolution of ransomware. (6 Aug. 2015). Retrieved July 9, 2018 from http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf
[16] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. Avclass: A tool for massive malware labeling. In *Int. Symp. on Research in Attacks, Intrusions, and Defenses.* Springer, Cham, 230–253.
[17] Alexander Sevtsov. 2017. Ransomware Network Communication. (14 July 2017). Retrieved July 9, 2017 from https://www.lastline.com/labsblog/ransomware-network-communication/
[18] Sara Tilly. 2017. Cryptolocker Prevention – How to secure your server environment. (29 March 2017). Retrieved July 9, 2018 from https://blog.syskit.com/cryptolocker-prevention
[19] Carl Woodward and Raj Samani. 2017. Is WannaCry Really Ransomware? (8 June 2017). Retrieved July 9, 2018 from https://securingtomorrow.mcafee.com/executive-perspectives/wannacry-really-ransomware/
[20] Michael Young and Ryan Zisk. 2017. Decrypting the NegozI Ransomware. (22 Sept. 2017). Retrieved June 07, 2018 from https://yrz.io/decrypting-the-negozi-ransomware