COMPUTER SCIENCE | RESEARCH ARTICLE

Cryptography and randomization to dispose of data and boost system security

Shaikhah B. Alkhadhr[1*] and Mohammad A. Alkandari[1]

Received: 05 December 2016
Accepted: 22 February 2017
First Published: 07 March 2017

*Corresponding author: Shaikhah B. Alkhadhr, Computer Engineering Department, Kuwait University, Kuwait City, Kuwait
E-mail: s.alkhuder@ku.edu.kw

Reviewing editor:
Tao Song, China University of Petroleum Huadong - Qingdao Campus, China

Additional information is available at the end of the article

Abstract: Massive public pressure is arising to achieve data privacy and protection. One way to maintain data privacy is to perform efficient data disposal processes. This minimizes the chances of data leakage over an extended period and through irresponsible actions. That is why adequate data disposal is essential in so many governmental and critical institutions. Simply “deleting” data does not erase it. In fact, it only removes the name that refers to that piece of data, but the data itself remains. In this paper, we utilize an encryption algorithm and then perform a randomization process to dispose of data values in an irreversible approach making it difficult to retrieve the original value of data from the defaced result.

Subjects: Computer Theory; Computer Science; Legal, Ethical & Social Aspects of IT; Computer Fraud, Hacking & Viruses; IT Security; Computer Engineering; Computer Science (General); Computer & Software Engineering

Keywords: data protection; privacy; data disposal; encryption; security; secret; automation; digital; application; deleting

1. Introduction and related work

Produced data on different computing systems are not equally important or required. Data are meant to be used for a specified period and for a cause, but when they are no longer of significant value, typically, they should be disposed of, which is not always the case of practice. It is a matter of user comfort and security when it comes for using websites, applications, or any typical device. Federal agencies, government organizations, corporates, social networks, e-commerce websites produce data waste regularly (Jedrzejczyk et al., 2009). If, however, these “outdated” information was not discarded adequately, it could lead to an unintended data leakage. Confidential data which is kept on different types of media need to be disposed of after their validity period has expired.

Shaikhah B. Alkhadhr

ABOUT THE AUTHORS

Engr. Shaikhah B. Alkhadhr and Dr Mohammad A. Alkandari have started research on information privacy and data protection as part of a master program class. This direction of research focuses on approaches of securing data sharing and storage, protecting them against illegal and unethical exploitation. Part of that process is studying the methods of data disposal and creating an optimized tool upon the findings of this study. This is the highlight of the currently presented research paper.

PUBLIC INTEREST STATEMENT

Privacy in the digital world has become scarce and practically rare to achieve. With the many digital ways, we use to convey our messages, express ourselves, or to perform our duties, our digital footprints are left almost in every website we visit, every hard-drive we use, and maybe with each keyboard strike we make. A “good deletion” method will help destroy the data that is expired, needed no more, or simply left in the wrong place. Unlike the traditional deletion method, this proposed method using efficient information security tools can assist performing a data deletion method that is irreversible; making sure that what we erase is truly deleted.

Educating users and IT staff about how to properly dispose of unused personal or sensitive data is one of the approaches to conserve privacy. It is a sign of a good system to have a healthy data disposal mechanism. Data privacy concerns stretch out to other fields besides governmental organization and businesses. These concerns also exist in academic environments, healthcare institutions, and others as the examples will describe later in this paper.

This paper contributes to:

(1) Emphasizing the threats of neglecting adequate data disposal methods on privacy and data protection.
(2) Advocating the need of having a customized approach for critical data disposal through introducing an algorithm that utilizes cryptography and randomization to process unwanted bits.
(3) Considering the possibility of implementing the algorithm in real operating systems like Linux based environments.

The rest of this paper is organized as follows: The remainder of this section covers related work and statistics of data disposal practices. Section 2 discusses the method and algorithm proposed. Section 3 studies the result obtained from executing different runs of the code. Section 4 summarizes the conclusion of the previous section results and the possible modifications to the algorithm variables, and finally Section 5 shows the future work to this approach and the potential utilization and integration chances.

Data destruction or sanitization (Srinivasan, 2012) can either be destructive or non-destructive. If it is destructive, then it includes physically destroying the unit. If it is non-destructive, then it is a process like overwriting or degaussing where the unit is preserved, but the data it holds is erased.

Approaching the end of 2016, Seagate had already shipped an average of 1.7 terabytes of hard drives (Seagate Technology LLC., n.d.). Based on a previous study (Srinivasan, 2012), it will take an approximate of 42 h to overwrite that amount of data. If you have access to the physical layer of a medium, you can destroy it to make sure the data it once carried is irretrievable, but if you only have user layer access to the medium, you are obliged to use the available interface to perform a deletion. However, unless you are aware of the exact block that holds the data you want to erase, it is almost impossible to overwrite data to insure it is not accessible after a simple delete. That is why the further you are from the medium, the harder it is to assert data destruction (Reardon, Basin, & Capkun, 2013). In the end, it is linked to the adversary sophistication, and how well are we ready to dispose of data.

Privacy is a major concern for individuals. Per the statistics by Teltzrow and Kobsa (Teltzrow & Kobsa, 2004), approximately 82% of users online have declined giving out their personal information and 34% have lied when answering questions regarding their personal lives. These numbers can indicate a fair amount of anxiety that people express when it comes to their digital privacy. Decades of inadequate data disposal can cause uncompensated data leakage either by intent or mistake like multiple SSN's, credit card and other personal information leakage. This has made illegal trading of user information easier to be involved in. There have been several federal laws warning individuals and organizations of such illegal actions, forcing penalties and fines to whoever is held responsible. Some of those are: Health Information Act (HIPAA), Personal Information Protection Act (PIPEDA), Gramm-Leach-Bliley Act (GLBA), California Senate Bill 1386, Sarbanes-Oxley Act (SBA) (Hughes, 2006).

A large amount of data can be collected over the internet from wide range of users with different characteristics and various behaviors (Birnbaum, 2004). The following highlights some examples of data that are used for a specific time and purpose, and then may be subject for irresponsible disposal methods (Rothke, 2009):

- Account records
- Activity sheets
- Applications
- Bank statements
- Bids and quotes
- Budgets
- Business plans
- Canceled checks
- Client lists
- Contact lists
- Corporate tax records
- General service information
- Health and safety reports
- Internal reports
- Magnetic media
- Personnel files
- Test scores/class rosters
- Encryption key management information

Because so many of the previous data examples are scattered all over the network and physical media, either from previous data-mining efforts or simply from huge data collection forms and loggers, they become hard to minimize, that is, if it wasn't already too late, when the data has been handled as merchandise between data brokers. So many chunks of data are being left behind, forgotten, or ignored intentionally. Confidential personal data are just lying around somewhere until someone takes the effort and time to search for them in the right place. Once found or retrieved from bad disposal methods, they can be used for illegal/unethical activities.

Hard copies of data should be physically destroyed towards the aim of minimizing data retrieval chances. US laws had made it official and obligatory to securely dispose of such data (Hughes, Coughlin, & Commins, 2009).

In typical scenarios in the Arabian gulf area, the process of recycling handheld devices demand secure and efficient personal data disposal to protect privacy. So many individuals lack the appropriate knowledge to dispose of the information on their devices when they decide to sell their old smartphones or give them up for recycling. With the right recovery tools, data residing there can be extracted and processed elsewhere. It can even be used in abusing activities.

With so many different methods available for data disposal, besides the normal delete command, we will see that different applications require different data disposal processes that vary in strength. Resources, effort, and time that organizations or individuals are willing to spend into the process of data disposal, and the incentives of doing so, are all factors that help decide which is the most appropriate data disposal mechanism for the concerned system.

Different research efforts were dedicated to recover data when corrupted concluding that it is relatively hard to accomplish full recovery out of complicated sanitization processes (Candes & Tao, 2005).

Research in the field of data disposal show various terms and methods for data disposal processes. An example of such is "Secure Sanitization" defined as the "erasure of both pointers and file data"

(Hughes et al., 2009). For computer hard drives, secure sanitization is to have the hardware itself destructed. So basically, data sanitization, is a term used by NIST to refer to all data elimination methods, "including block-by-block over-write, drive internal secure erase (SE), and physical, chemical, thermal, or magnetic destruction" as the IEEE Security and Privacy journal stated (Hughes et al., 2009). Garfinkel and Shelat explain the types of data and their recovery difficulty (Garfinkel & Shelat, 2003):

- Level 0: Regular files that can be easily found in C\Windows directory.
- Level 1: Temporary files such as the browser cache, helper files, or recycle bin files, where the user expects the system to delete them.
- Level 2: Deleted files, where the files are removed from the file system but their blocks are not overwritten, and the files references are still in the containing directory. Deleted files are simply tagged "free". Traditionally recovered using recovery tools.
- Level 3: Retained data blocks which are data blocks recovered but not belonging to a named file such as slack space or virtual memory backing store. They can also simply be level 2 data partially overwritten.
- Level 4: Vendor-hidden data that can only be fetched with vendor-specific commands. Normally used for drive management.
- Level 5: Overwritten data that further distance recovery of data.

1.1. Common practices of data destruction and possible retrieval methods

The data disposal is done either physically or electrically to avoid data leakage. Table 1 shows a few standards used by Cornell University (Cornell University, 2014). Media or data can suffer leakage for a number of reasons such as:

- PC's that have been sold or disposed of without wiping the resident data.
- Most of sellers take the damaged hard disks and repair them, which then place data in a position accessible again.

There are standards for media disposal such as DOD 5220.22 and NIST 800-88. The process of DOD 5220.22 is to overwrite the existing data 3 times on the functional drivers to ensure original data is no longer recoverable. This protocol is also used to insure asset readiness for reuse. The security office in Cornell Security Department recommends DOD 5220.22 to be applied in case of reusing hardware parts.

Gutmann (1996) argues that there are still some ways to read the overwritten data. One of these is the magnetic microscopes. Although there has not been any practical proof to back his statement, however it is worth mentioning since it is within the scope of related works.

1.2. Hardware erasure

The term hardware here refers to disk, tape data, hard drives, or any other type of tangible mass storage piece. Normally, this is a process that can be done in one of multiple methods, and each method is evaluated by two parameters; security level and completion time (speed).

Table 1. Disposal methods

Object type	Disposal method
Hard disk	Physical destruction or degauss
Floppy disk	
Tapes	
Memory cartridge	
Caseless optical (CD/DVD)	Physical destruction
Small solid state, USB/Flash	

Security levels vary from very weak, in which data is easily recoverable, to extremely difficult or almost impossible, where data is nearly lost with extremely low recovery chances. As for the speed of each erasure process, Hughes and Commins comparison (Hughes et al., 2009) shows that it can be a period approximated in milliseconds or it could take up to a half of a day. Best performing method in both parameters is physical destruction.

1.3. Data disposal using encryption

Disposal trends are leaning more towards using encryption tools. These are methods where encryption key is used to handle drive requests like keeping or erasing data. Keys in this situation typically never leave the hard drive. Previously, this consumes 30 to 60 min of time, but now, secure discarding of data is simply done by changing the encryption key which only takes up to a few microseconds. Yet, of course, this operation can only be done by the individual aware of the original master key; publicly known as the password. Some of the known platforms that provide this is Windows 2000, and XP, yet, it all gets back to the password. Disclosing it causes insecurity (Hughes, 2006).

TPM or the trusted platform module is a method used to generate and manage keys using random number generators. This can replace the need of using the password in encryption operations. Typically, it should provide security to different disk locations, computational requirements change per the level of security demanded. This serves to make recovery of original data impractical, or in other words, completely disposed of (Melvin, 2010). Again, this is related to a passphrase that can be easily forgotten by the user causing unintentional data disposal.

2. Method

The effort exhibited in this research concentrates on assuring that data will be hard to retrieve. It is targeting the values of data that are highly sensitive and are targeted for disposal. Using the simple combination of crypto-techniques, an elementary function that disturbs the encrypted result, and a direct language encoding to accomplish data disposal are the three main factors to perform this disposal method. This is based upon the assumption that ciphered data which has been corrupted is difficult to revert to its original form. The algorithm receives input as plain text or broken down data strings, it then takes it through a series of stages, and finally represents some unrecognized output; all based on the input and three defined algorithms. The proposed approach consists of three main stages; Encryption, corruption, and decryption. Minor stages can be added if required, like a preceding stage of breaking data into blocks of compatible size with the encryption algorithm used, or a superseding stage of overwriting the original data with the result repeatedly as an assurance for an irreversible process. The three main stages can be summarized as in Figure 1.

Human written passwords cannot be considered as encryption keys in this approach. Keys should have more complex properties that minimize the chances of having them regenerated (guessed) or easily forgotten. Involving encryption keys in the process of data disposal is to complicate the process of original form recovery. This leaves the user with blocks of defaced data present on the storage structure, unresolvable, and ready to be overwritten if necessary. To illustrate this attempt of data disposal approach, an executable code was designed and implemented.

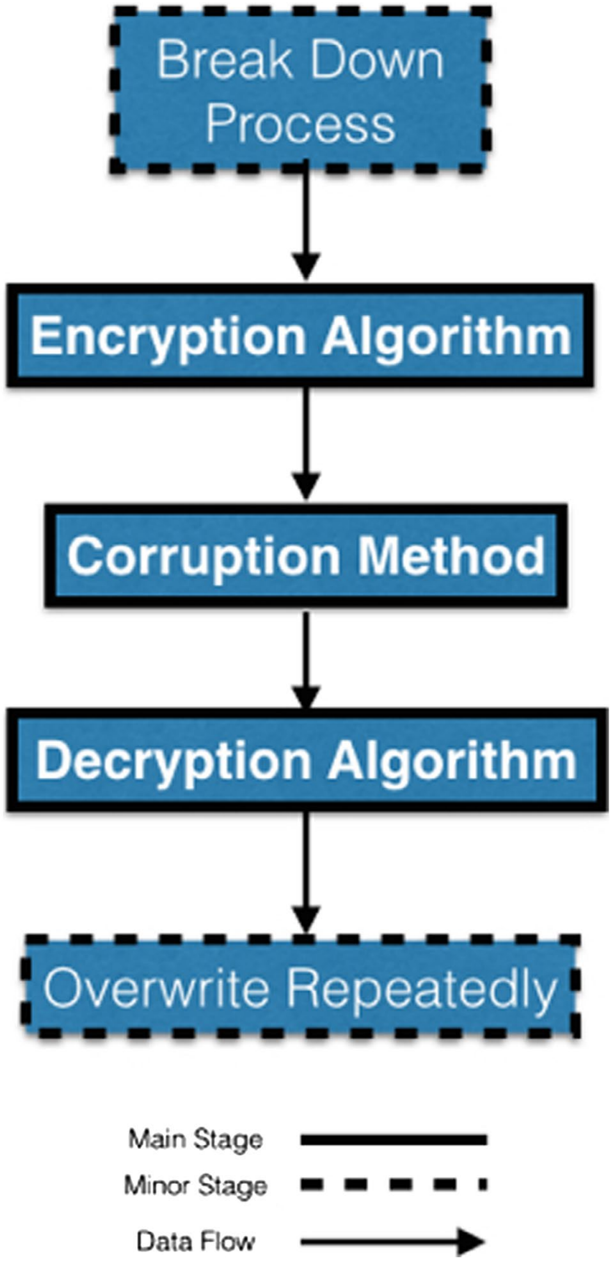
2.1. Environment of development

The development environment used for testing the proposed approach is a common MAC OS 10.10.1 (Yosmite). The implementation tool used for developing the code is Xcode version 6.1. Coders using Xcode as a development tool, may have already noticed that attempting to use the standard programming language of Xcode version 6.1, SWIFT, to emphasize a security aspect in a certain application is relatively risky. SWIFT is a moderately new language introduced, so it does contain a standalone AES-256 function to utilize in various implementations.

2.2. AES-256 bit secret key level

The first stage is to execute an encryption algorithm. This is done to obtain the cipher result of the original text to be disposed of. This stage can be any encryption algorithm that delivers a cipher

Figure 1. Sequence of stages in disposal process.



result possible to tamper with. Some of the common encryption algorithms that can be used are DES, 3DES, AES, blowfish, PGP … etc. The encryption algorithm utilized here is AES with a secret key size of 256 bits. In this approach, if data input is larger than 128 bit, it needs to be broken down to that size, and padded whenever the need requires to keep input block size 128 bits.

The advanced encryption standard (AES) is a common encryption algorithm used recently in various applications from governmental to commercial ones. This encryption standard takes in data broken down to blocks of 128 bits. The security level varies using the same standard but different secret key sizes. Secret key sizes can be 128, 192, or 256 bits with each offering a different level of security. This encryption standard does 14 rounds with 8 S-boxes to perform the encryption process (Biryukov & Khovratovich, 2009).

AES is chosen to be a part of this data disposal method due to the low number of successful cyber-attacks announced against it. It is hard to accomplish a successful attack against this encryption standard specially if the key size used is large. This means that an adversary cannot duplicate the result produced by this encryption algorithm. AES performs more efficiently in terms of hardware and software with relatively little key setup time and memory requirements as opposed to other algorithms such as blowfish which is best suited for software efficiency, and requires longer key setup time and larger memory requirements, however, it does work steadily faster after the key setup phase. Blowfish encrypts smaller block sizes than AES; 64-bit block size in comparison to 128-bit block sizes of AES. Since the key size used in Blowfish can go up to 448 bits, then the security level for it can surpass that of the 256-bit key of AES algorithm, therefore it is possible to use Blowfish in the suggested approach if it is applied in environments where memory resources are less limited (Hamburg, 2014). Because AES is a widely-used algorithm, it was easy to find a simplified ready-to-use segment of code to include in this implementation (Biryukov & Khovratovich, 2009).

All strings in the code are transformed to UTF8 strings for better processing and then are turned to base64 strings for better representation. The already composed code is referenced with comments within the Xcode project. Comment lines at the beginning of each class refer to the original composers of the code. A significant deal of customization is done to the Obj-C project classes: "CryptTestViewController.h", and "CryptTestViewController.m" (Takeuchi, 2010). Other code classes are concerned with generating keys and byte form representation. The following code segment is used to implement the encryption process. The main reason of performing the conversions between ASCII, base64, and UTF8Strings is to bring about a better representation of output results for the user to comprehend and identify. The detailed definition of the encryption algorithm is kept abstract:

```
- (IBAction)Encrypt:(UIButton *)sender {

str=_PlainText.text;

// 1) Encrypt

NSLog(@"encrypting string = %@",str);

NSData *data = [str dataUsingEncoding: NSASCIIStringEncoding];

NSData *encryptedData = [data AESEncryptWithPassphrase:password];

// 2) Encode Base 64

[Base64 initialize];

NSString *b64EncStr = [Base64 encode:encryptedData];

NSLog(@"Base 64 encoded = %@",b64EncStr);

_EncryptedText.text = b64EncStr;

}
```

The command NSLog was originally used to track the values of the resulted string across the different stages without the need to print values to a GUI window.

2.3. How to corrupt a ciphered result?
Number of randomization passes differ according to data sensitivity. It is a debatable variable that can be used to enhance the performance of the algorithm since it is directly linked to the number of read and writes performed on memory blocks. Overwriting can be done through wiping zeros or random patterns which we use in our implementation. This makes it hard to tell if there were any original data on the medium (Srinivasan, 2012).

Corruption of data is basically deciding to tamper with the bits in the cipher result we obtained in this stage. Randomization of the corruption process has appeared best to divert the result from its original form.

In the corruption stage, a method of letter-case shifting (Stack Overflow, n.d.) was chosen. The method basically chose random letters of the input string and then it would shift the case of these chosen letters. If the letter was a lower-case letter it would shift to be an upper-case letter, and vice versa. This has been chosen because it is a method that utilizes the randomness factor, and it takes advantage of the gap between upper and lower case enumerated representation in ASCII. The following code segment has been used to do this task:

```
- (IBAction)CorruptCipher:(UIButton *)sender {

NSString *b64EncStr = _EncryptedText.text;

NSMutableString *CorruptedTxt = [NSMutableString string]; // to be able to manipulate
characters separately

for (NSUInteger i = 0; i < [b64EncStr length]; i++)

{

NSString *substring = [b64EncStr substringWithRange:NSMakeRange(i, 1)];

[CorruptedTxt appendString:(rand() % 2) ? [substring lowercaseString] // %2 is used to get
results in range of 0 -> 1

        : [substring uppercaseString]]; //rebuilds the cipher text after randomly shifting its
letters to upper and lower cases

        }

NSLog(@“corrupted string = %@”,CorruptedTxt);

_EncryptedText.text = CorruptedTxt; //show the randomized cipher text

}
```

The class NSMutableString is used to grant the ability of tampering with the string contents (Apple Inc, n.d.). It is noticed that the main variables that show the output string are pointed at using “*”, meaning that the change is being applied directly to the variable’s value through a variable pointer instead of a value copy.

2.4. Decryption of data
Decryption is used in a straighter forward manner. The stage of decryption takes any string resulting of the corruption function and simply decrypts it to a defaced form because of the corruption method previously performed. After this point of the proposed approach, the result is something

practically hard to predict. The following shows the process of decoding the corrupted cipher. Follows that is the decryption process of the corrupted string.

```
- (IBAction)Decrypt:(UIButton *)sender {

// 3) Decode Base 64

NSString *b64EncStr = _EncryptedText.text;

NSData *b64DecData = [Base64 decode:b64EncStr];

// 4) Decrypt

// This should be same before encode -> decode base 64

//NSData *decryptedData = [encryptedData AESDecryptWithPassphrase:password];

NSData *decryptedData = [b64DecData AESDecryptWithPassphrase:password];

NSString* decryptedStr = [[NSString alloc] initWithData:decryptedData encoding:NSASCIIStri
ngEncoding];

NSLog(@"decrypted string = %@",decryptedStr);

_DecryptedText.text = decryptedStr;

}
```

The decryption algorithm shows a different result for each different corruption output. This GUI interface also allows the user to modify the cipher result manually for testing several results.

The original code does not show the details of variable contents anywhere else besides the NSLog console. To be able manipulate the cipher text or tamper with any results prior to the decryption process we need to acquire a UI for better accessibility to variable contents. We then simulate the encryption, corruption, and decryption stages using a window with three UIButtons and three text fields.

Each of the three buttons, when pressed, should perform the functionality in its title. An IBAction is defined for each UIButton separately. The IBAction method definition is the procedure of converting the encoding type and executing functions of encryption, decryption, and corruption.

2.5. Clear data and text holders

The simulation window is composed of UITextFields and UIButtons. On each press, one of the three functions are called to either encrypt the string input, corrupt the cipher text, or decrypt the corrupted string. The simulation runs in a discontinuous pattern. This is done to enable exploring manual and programmed corruption methods, and to analyze different results and program flows that show on the NSLog console.

With all the previously discussed GUI elements that allow user input, a clear button is added to flush input values and prepare the algorithm for a new run. A direct clear function is implemented as follows:

```
- (IBAction)clearTexts:(UIButton *) sender {

_EncryptedText.text=@"";

_DecryptedText.text=@"";

_PlainText.text=@"";

}

- (void)dealloc {

[_PlainText release];

[_EncryptedText release];

[_DecryptedText release];

[super dealloc];

}
```

The function clears all text values on the GUI window and deallocates all memory space that was originally preserved to keep the user input values. This functionality basically prepares the text fields for any new values to be processed.

3. Results and discussion

3.1. Assumptions

To start analyzing the proposed approach, we assume the following:

- This approach is applied to file types defined by the user, such as jpeg, txt, pdf … etc., whichever indicates the most important to the user. This applies the algorithm to part of the data on a structure instead of brute force overwriting all data.
- All data can be broken down to the required block size.
- All data blocks are available in plain form or in encrypted form (if in the second form, the first main stage of the approach can be ignored).
- Any encryption algorithm can be used. The more advanced the algorithm, the more it increases the chance of result irreversibility. This is due to the difficulty of regenerating a similar result without the encryption key. The algorithm applied in this case is AES with key size: 256 bits.
- Any corruption algorithm can be used. The more random it is done, the higher the chances of result irreversibility. Corruption algorithm applied in this approach is shifting letter cases at random indexes of the string. Although it is possible to perform text corruption manually, it is not practical.
- The final minor stage of overwriting the result can be done manually or can be set to be automated with several passes if required.

3.2. Test-running the encryption algorithm

For the first run of the implementation, we assure that the encryption algorithm is working as it should. So, the input in the plain text field is set to a simple clear text value like: "Hello World!".

The encryption function is then executed. A cipher text will appear in the encrypted text field. The Decryption function is summoned after that to assert that cryptographic process is working properly. Result should appear in the log console as follows:

```
Dispose[11416:808875] encrypting string = Hello World!

Dispose[11416:808875] Base 64 encoded = iZT9E50v/yJyZ+kYMzfj/A==

Dispose[11416:808875] decrypted string = Hello World!
```

After assuring the functionality of the encryption algorithm, analyzing the various results of different corruption cases can take place.

In the early stages of implementing this approach, corruption of the encrypted text was done in an ad hoc fashion, changing random letters; this can be done by the user, deleting part of the cipher or adding a few base64 characters to recreate a sort of corruption. This was not sufficient, so an algorithm with some sort of corruption process (randomization) was required.

3.3. Automating the corruption process

The corruption of data must be as random as possible. If an adversary notices the pattern of change in the cipher text, it is only a matter of time until a successful recovery process happens. This is possible for applications where there exist repeated data like for instance bank system logs; some of the repeatedly occurring commands are deposit or withdraw. For a database with hundreds of transactions categorized as either deposit or withdraw. Hence, encrypting either of these commands and running a script to modify the three last characters of the cipher form is obvious to detect.

To enhance the corruption process, a string manipulation function is implemented. This function, as stated previously, shifts letter cases randomly. NSLog console shows corrupted text after calling the CorruptCipher function as follows:

```
Dispose[11801:848216] encrypting string = Hello World!

Dispose[11801:848216] Base 64 encoded = iZT9E50v/yJyZ+kYMzfj/A==

Dispose[11801:848216] corrupted string = izt9E50 V/yJyZ+kymzFJ/a==

Dispose[11801:848216] decrypted string = uZ24>¥ÂÕ?HpY
```

4. Conclusion

4.1. Possible integration in operating systems

With the help of a detection algorithm, this approach can be integrated within an operating system to identify the types of files and perform the disposal algorithm accordingly. Recognizing files that should be deleted instead of the whole disk files can save up time and resources. If a user concerned more about jpeg files then this algorithm can target those types of files, adequately destroy them instead of going through all files that might hold common insensitive information and the total opposite in close rates.

This algorithm can be integrated in operating systems in such a way that it can be called by executing a command other than the typical delete or erase/format. This is done to protect against any malicious activity that might occur in an infected machine such as duplicating files prior to any deletion command to prevent disposal.

4.2. Number of passes that the algorithm should perform

As the results show, Decrypting corrupted cipher data products garbage values. If the corruption algorithm is run more than one time, it continues to manipulate the current input. However, corrupting a cipher text *n* times does not necessarily decrease the chances of recovery. Multiple cipher text tampering becomes more of a normal edit to a garbage value.

It is already impossible to predict what random value does the seed inputs to the randomized manipulation algorithm so repeating the process of corruption does not necessarily show that repetition will boost the efficiency of the disposal algorithm. One time pass of the algorithm already results in a highly irreversible value, given that the encryption cannot be undone without a key, and even if the key is compromised, the encrypted result is manipulated half way in the algorithm, which gives a successful encryption process no significance in terms of retrieving the original data.

5. Future work

Randomization is an important factor to consider when attempting to destruct data or reuse resources. Although utilizing encryption techniques to dispose of data affects the cost and runtime for large amounts of data, it can be a reliable and efficient method for data-disposal. Data pieces that are relatively small and critical in value which does not bare to be left as digital residue can be disposed of using this approach. This widens the scope of cryptographic tools applications. In the future, this approach can be enhanced to be able to identify sensitive data using machine learning and pattern recognition.

Acknowledgements
Special thanks for Professor Tassos Dimitriou for enriching the topic and emphasizing the right literature subjects to review. Great appreciation to Cyrus, Kiichi Takeuchi, and Jeff LaMarche for provide the online source code donations for base64, AES, and Rijndael as open source classes for other developers. Thanks is also dedicated for the “Stack Overflow” community for providing useful programming discussions and troubleshooting assistance.

Funding
The authors received no direct funding for this research.

Author details
Shaikhah B. Alkhadhr[1]
E-mail: s.alkhuder@ku.edu.kw
ORCID ID: http://orcid.org/0000-0001-5938-2953
Mohammad A. Alkandari[1]
E-mail: m.kandari@ku.edu.kw
ORCID ID: http://orcid.org/0000-0002-0893-6116
[1] Computer Engineering Department, Kuwait University, Kuwait City, Kuwait.

Citation information
Cite this article as: Cryptography and randomization to dispose of data and boost system security, Shaikhah B. Alkhadhr & Mohammad A. Alkandari, *Cogent Engineering* (2017), 4: 1300049.

References

Apple Inc. (n.d.). *NSMutableString*. Retrieved November 15, 2016, from https://developer.apple.com/reference/foundation/nsmutablestring

Birnbaum, M. H. (2004). Human research and data collection via the internet. *Annual Review of Psychology, 55*, 803–832. doi:10.1146/annurev.psych.55.090902.141601

Biryukov, A., & Khovratovich, D. (2009). Related-key cryptanalysis of the full AES-192 and AES-256. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (vol. 5912 LNCS, pp. 1–18). doi:10.1007/978-3-642-10366-7_1

Candes, E. J., & Tao, T. (2005). Decoding by linear programming. *IEEE Transactions on Information Theory, 51*, 4203–4215. doi:10.1109/TIT.2005.858979

Cornell University, Best Practices for Media Destruction. (2014, September 12). Retrieved September, 2015, from http://www.it.cornell.edu/security/how.cfm?cat=6&tip=57

Garfinkel, S. L., & Shelat, A. (2003). Remembrance of data passed: A study of disk sanitization practices. *IEEE Security and Privacy, 1*, 17–27. doi:10.1109/MSECP.2003.1176992

Gutmann, P. (1996). Secure deletion of data from magnetic and solid-state memory. In *Sixth USENIX security symposium proceedings* (pp. 77–90). Retrieved from https://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html\nhttp://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html

Hamburg, M. (2014, April 7). *Cryptography: What is the difference between AES, blowfish, and DES?* Retrieved February 3, 2017, from Quora, https://www.quora.com/Cryptography-What-is-the-difference-between-AES-Blowfish-and-DES

Hughes, G. (2006, October). *Security for disk drive data at rest … disk drive opportunities?* Retrieved October, 2015, from http://www.media-tech.net/fileadmin/templates/resources/sc06/mtc06_keynote_day1_hughes.pdf

Hughes, G. F., Coughlin, T., & Commins, D. M. (2009). Disposal of disk and tape data by secure sanitization. *IEEE Security and Privacy, 7*, 29–34. doi:10.1109/MSP.2009.89

Jedrzejczyk, L., Price, B. A., Bandara, A. K., Nuseibeh, B., Hall, W., & Keynes, M. (2009). *I know what you did last summer: Risks of location data leakage in mobile and social computing*. Retrieved from http://computing-reports.open.ac.uk/2009/TR2009-11.pdf

Melvin, S. (2010, September). *The forgetful disk drive—Zytek communications corporation*. Retrieved October, 2015, from http://www.zytek.com/ForgetfulDiskDrive.pdf

Reardon, J., Basin, D., & Capkun, S. (2013). SoK: Secure data deletion. In *Proceedings - IEEE Symposium on Security and Privacy* (pp. 301–315). doi:10.1109/SP.2013.28

Rothke, B. (2009). *Why information must be destroyed*. Retrieved November 14, 2016, from http://www.csoonline.com/article/2123705/privacy/why-information-must-be-destroyed.html

Seagate Technology LLC., Seagate Technology Announces Preliminary Financial Information For Fiscal Fourth Quarter And Year-End 2016. (n.d.). Retrieved November 15, 2016, from http://www.seagate.com/em/en/about-seagate/news/seagate-technology-announces-preliminary-financial-information-for-fiscal-fourth-quarter-and-year-end-2016-pr/

Srinivasan, A. (2012). ERASE—Entropy-based robust sanitization of sensitive data. In *The 7th international conference for internet technology and secured transactions* (vol. 1998, pp. 427–432).

Stack Overflow, Random Uppercase - Lowercase. (n.d.). Retrieved November 15, 2016, from http://stackoverflow.com/questions/7060154/random-uppercase-lowercase

Takeuchi, K. (2010, April). *Github: AES-256 basic crypt test algorithm for Xcode*. Retrieved December, 2014, from https://github.com/ulhas/crypt/find/master

Teltzrow, M., & Kobsa, A. (2004). Impacts of user privacy preferences on personalized systems. *Designing Personalized User Experiences in ECommerce Human-Computer Interaction Series, 315–332*. doi: http://dx.doi.org/10.1007/1-4020-2148-8_17