ON EFFICIENCY OF DISTRIBUTED PASSWORD RECOVERY

Radek Hranický Martin Holkovič Petr Matoušek Ondřej Ryšavý
Brno University of Technology,
Božetěchova 2, Brno, Czech Republic
{ihranicky, iholkovic, matousp, rysavy}@fit.vutbr.cz

ABSTRACT

One of the major challenges in digital forensics today is data encryption. Due to the leaked information about unlawful sniffing, many users decided to protect their data by encryption. In case of criminal activities, forensic experts are challenged how to decipher suspect's data that are subject to investigation. A common method how to overcome password-based protection is a brute force password recovery using GPU-accelerated hardware. This approach seems to be expensive. This paper presents an alternative approach using task distribution based on BOINC platform. The cost, time and energy efficiency of this approach is discussed and compared to the GPU-based solution.

Keywords: password recovery, distributed computing, digital forensics, BOINC, cost efficiency

1. INTRODUCTION

In the area of digital forensics, the use of password protection presents a great challenge for investigators while conducting examinations where documents and files of a suspect are encrypted. Common user applications like MS Office, PDF creator or archive programs offer the password-based protection of the content using encryption by AES or RC4 algorithms.

To overcome this kind of protection, password recovery tools like John the Ripper[1], Elcomsoft tools[2], oclHashcat[3] or Wrathion[4] can be employed. These tools are designed to run on a single CPU hardware, or in GPU-accelerated mode.

Several studies showed that for shorter alphabets (up to 26 characters) and passwords shorter than 8 characters, the password can be broken within tens of minutes on common CPU-based hardware (Hranický, Matoušek, Ryšavý, & Veselý, 2016). When GPU-accelerated hardware is used, recovery time drops even to minutes. For passwords longer than 8 characters, the recovery process on one CPU is unfeasible. GPU approach seems to be a nice solution for forensic experts, however, there are several limitations. First, GPU cards are vendor dependent, i.e., password recovery software must be optimized and compiled for a specific GPU hardware. Second, multi-GPU approach requires a specially tailored computer with optimized power and additional cooling which increases the total cost of the solution. Third, some password recovery tools do not support GPU acceleration.

There is also another approach to password recovery based on distributed computing on multiple nodes that participate in exhaustive search of possible passwords, aka brute force attack. Recovery of passwords protected by one-way hash function can be effectively parallelized and distributed. Comparing to a single CPU or GPU version, several issues must be addressed: When is the distributed approach more efficient than GPU-based approach? How to effectively distributed the computing in heterogeneous network of nodes with different performance?

[1]See http://www.openwall.com/john/
[2]See https://www.elcomsoft.com/products.html
[3]See http://hashcat.net/oclhashcat/
[4]See http://wrathion.fit.vutbr.cz

In this paper, we present a dynamic architecture for distributed password recovery based on an open source platform BOINC[4]. The first results show that distributed solution can be a cost efficient and modular alternative for solving password recovery on common hardware. The main advantage of this approach is to use a just-available hardware for computing without any additional costs for high-performance GPU hardware. In addition, this approach provides easily scalable and hardware independent solution.

1.1 Contribution

The paper discusses how password recovery can be solved in distributed environment. There are many practical issues to be considered: the efficient task distribution, communication overhead, or dynamic behavior of the cluster where nodes join and leave during the computing.

The paper proposes an efficient and scalable architecture that is independent on hardware, number of nodes, encryption algorithms, or type of password generators. The architecture is built upon BOINC communication platform and employs an extended open-source recovery tool Wrathion (Hranický et al., 2016). The proposed solution is freely available at http://wrathion.fit.vutbr.cz.

Efficiency of this approach is evaluated with respect to the number of connected nodes and the password length. In comparison to GPU cracking, the proposed architecture offers a scalable and low cost solution for password recovery.

To preserve privacy of recovered documents, first meta data needed for password recovery are extracted of the document. Only meta data are transmitted to the working nodes what increases privacy and decreases communication overhead.

A key aspect of this solution is an efficient task distribution based on the adaptive time estimation that reflects elapsed time of the entire job, the number of active nodes, and the performance of a node. The algorithm is presented here.

1.2 Structure of the Paper

The paper is structured as follows. Section 2 presents related works in the area of distributed password recovery. In Section 3, the distributed approach for password recovery is discussed. Section 4 introduces our solution. Experimental results and comparison to GPU-based approach is shown in Section 5. The last section concludes the work and discusses its practical deployment.

2. RELATED WORK

The issue of password strength and password recovery using analytical methods, dictionaries, brute force attack, or heuristic approaches has been extensively studied. Useful observations concerning password strength and password cracking can be found in (Kelley et al., 2012; Ur et al., 2012; Reimann, 2013) or (Dell'Amico, Michiardi, & Roudier, 2009). There are also many papers focused on analytical and heuristic approaches like rainbow tables (Thing & Ying, 2009), Markov chains (Marechal, 2007), etc.

In our approach, we limit ourselves to brute force password recovery and its efficiency in distributed environment in terms of costs and scalability. Brute force attack (exhaustive search) is effective if passwords are randomly generated which limits the usage of dictionaries or Markov chains. The usage of rainbow tables is also unfeasible for longer passwords. The rainbow table usually contains a pre-computed pair of the input and output value for a given hash algorithm. The size of the table grows exponentially with the size of the character set used. For example, the rainbow table for SHA-1 algorithm, ASCII charset and passwords up to the length of 8 characters has the size about 460 GB[5]. In addition, many encryption algorithms require multiple iterations of hashing or adding the salt which makes the usage of rainbow tables too expensive.

Our focus is on effective distribution. One of the first works on distributed password cracking was introduced by (Pellicer, Pan, & Guo, 2004) where BOINC architecture was used to distribute MD4-algorithm cracking of five character passwords to four nodes. The authors used BOINC scheduling system for task distribution. They observed that the system did not consider

[4]See https://boinc.berkeley.edu/

[5]See http://project-rainbowcrack.com/table.htm

relation between the scheduling and the overall size of the password space. The improper scheduling together with the large amount of transmitted data caused a serious bottleneck of the system. In our approach, we eliminate this drawback using an adaptive scheduling mechanism. We also reduce the amount of transmitted data by sending meta data only.

(Apostal, Foerster, Chatterjee, & Desell, 2012) evaluated password recovery at the high performance computing (HPC) platform using Message Passing Interface (MPI). They divided password database file among MPI nodes which distributed the task to GPUs. While reaching high acceleration, they also encountered some limitations that affected the design and implementation of password recovery application, e.g., each GPU required its own controlling thread and memory context was lost whenever the threat exited. Also, transmission of large password files over network caused significant overhead.

Similar approach was also implemented by (Marks & Niewiadomska-Szynkiewicz, 2014) in their high performance distribution platform called HGCC for CPU and GPU-based password recovery. Unfortunately, they do not discuss the task distribution of the architecture what is crucial for efficient usage. Our approach is based on an open source BOINC platform where scheduling and task distribution are improved by qualified time estimation of the node performance.

3. DISTRIBUTED COMPUTING

The main goal of this works is to evaluated the efficiency of distributed password recovery in heterogeneous environment. This part discusses requirements and limits of distributed computing.

3.1 Requirements

First of all, the distributed password recovery requires a tool that provides password recovery on a single machine. This tool will be distributed together with required data to a set of workstations which creates *a high-performance virtual machine*. Communication platform should be able to control a variable number of computing nodes with different computational power where the performance can change over time.

Another requirement is a platform independence, i.e., working nodes can be workstations with MS Windows, Linux or Mac OS installed and that can be fully or partially involved in the distributed computing without further modification of the operating system or adding extra hardware. Upload of the application and the task distribution should be provided automatically. Since encrypted documents or archives can be large, e.g., GB ZIP-files, distribution of the entire file across the network is not effective. Assuming a central node and multiple working nodes, the central server should extract the required meta data first and then it should distribute them to each working node over the network without the need to share the whole document or file. Similarly, password space should be generated on working nodes during the recovery process rather than distributed.

3.2 Effective Task Distribution

Depending on the recovery method (brute force, dictionary, heuristics-based attack), and task definition (encrypted document type, algorithm, alphabet, password length), the central node determines the set of all possible passwords. This set is split into smaller subsets and assigned to each working node. There are two options how to distributed the task to working nodes: (i) *full distribution*, or (ii) *progressive distribution*.

In full distribution, each subset is assigned to a working node at the beginning of the recovery process. The number of subsets is equal to the number of nodes: the distribution is uniform. Communication overhead is minimal since each node receives all data before the processing. The full distribution is ideal for the fixed number of working nodes with the equal performance.

In environment of variable number of working nodes with different computational power, progressive distribution is more recommended. Scheduler assigns a small portions of the password space to each working node. Once the subset is processed, the next one is assigned *on-the-fly*. This approach increases utilization of working nodes. It requires a good planning strategy.

3.3 Working Nodes

A working node generates a subset of the password state space, computes a hash for each generated password and compares this hash with the hash of the encrypted document. If they match, the node reports the successful recovery to the central node. If not and the progressive task distribution is applied, the node asks the central node for a new subset of passwords to be tested.

The aim is the optimal usage of the computational power of all working nodes. Section 4 shows how this can be achieved.

3.4 Distributed Frameworks

For distributed password recovery, two popular frameworks were considered: *Message Passing Interface* (MPI)[6] and *Berkeley Open Infrastructure Network Computing* (BOINC)[7].

3.4.1 MPI Framework

MPI library provides an efficient way of coarsely dividing work between multiple computers, cores, or threads where each individual unit performs calculations on a subset of the data. As discussed in (Kang, Lee, & Lee, 2015), MPI shows great performance for moderate amount of data and computational-intensive problems as password recovery is. OpenMPI[8] is an implementation of MPI that includes various fault-tolerance techniques: local or distributed checkpoints, network failure detection, etc.

Original design of MPI does not consider dynamically added nodes. This functionality is supported from MPI version 2.0. Unfortunately, new nodes are not detected automatically, so extra process is needed to detect new connections.

MPI does not natively support encryption or authentication. Thus, security also needs to be implemented by an application programmer.

There are several open source (OpenMPI) and commercial solutions (Intel MPI).

3.4.2 BOINC Framework

Although MPI platform satisfies many of our requirements, we assumed that *Berkeley Open Infrastructure for Network Computing* (BOINC) is more suitable for our purposes. BOINC is a platform for distributed computing that natively provides dynamic number of nodes connected over the Internet (Anderson, 2004). It was developed by U.C. Berkeley Space Sciences Laboratory and is distributed as open source.

The platform is used for public-resource projects like SETI@home where volunteers participate on solving various problems. It is also suitable for grid computing.

Every computing problem is called *a project*. The problem is divided into a number of tasks. Once the project is created, clients can connect and participate on the solution by sharing their computational power. The project server creates and schedules the tasks, keeps track of clients, maintains data storages, etc.

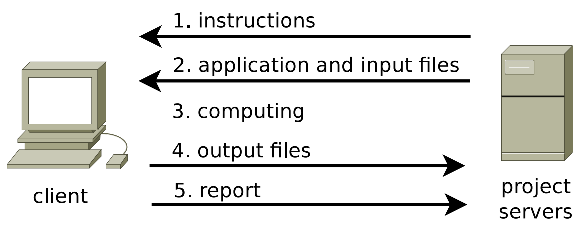


Figure 1: Solving a single task using BOINC

The task is a smallest piece of work within the project. The client can solve one or more tasks at time. Figure 1 illustrates the communication between the project server and the client to solve a single task. At first, the client receives instructions describing the task. Task assignment is client-specific, i.e., client architecture, operating system (OS), and hardware are taken into account. Thus, BOINC allows the scheduler to create a task tailored to client's configuration.

For solving the task, a client may need one or more applications. BOINC supports automatic distribution of binaries with respect to client's architecture. It also distributes input data.

Once the client receives all necessary data, it starts the computing which takes minutes, hours, or days. When the client completes its task, the output files and the task report are sent back to the server. Then, the client waits for a next task.

Unlike MPI, BOINC is natively designed to

[6]See http://www.open-mpi.org
[7]See https://boinc.berkeley.edu/
[8]See http://openmp.org

distribute tasks over the Internet. It provides built-in security for untrusted environment: authentication, user account management, digital signatures and public-key encryption.

Clients can dynamically connect and disconnect to a running project. They can even specify the percentage of CPU power to be assigned to the task, upload or download limits, disk and memory size utilization, etc. It is also possible to define at which time the computing can start.

4. THE PROPOSED ARCHITECTURE

A functional architecture of the BOINC-based solution is depicted in Figure 2.

4.1 Project Server

The project server is the main component of the proposed architecture. The task is created using the *User interface* (UI). Each task includes an encrypted document, scheduling options, client limits, specification of password generator, etc. The process of password recovery includes:

1. Upload of an encrypted document and creation of the project
2. Extraction of meta data needed for the password recovery, see Section 4.1.1.
3. Selection of a password generating method, see Section 4.1.2.
4. Task creation and scheduling based on nodes' real performance and adaptive time calculation as described in Section 4.1.3.
5. Distribution of new tasks to nodes that have finished their computing, see Section 4.1.4.
6. Saving the password when found.

The operator can monitor the whole process of password recovery, task distribution, load of the working nodes and the results using UI.

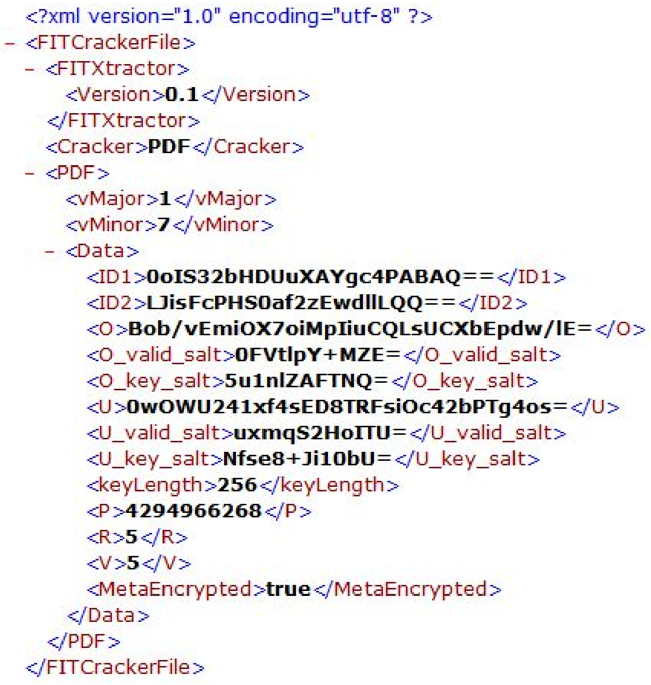
4.1.1 Extracting Meta Data

The uploaded encrypted document is processed by *FitXtractor* at first. FitXtractor is a preprocessor that identifies the type of the document based on a file signature, extension, or internal structure. If the format is supported, FitXtractor begins parsing its internal structure. The parsing is document-specific. For each type of the document, a new parser has to be implemented. Currently, FitXtractor supports MS Office Documents 97/2003, PDF, ZIP, RAR and MS Office XML formats.

The aim of the parsing is to collect all pieces of information needed for the password recovery. These meta data include document version, type of the encryption, cryptographic algorithms involved, key length, validation string, etc. For archives, compression type is added. Verification values in form of cryptographic hashes that will be compared with generated passwords form the essential part of meta data.

FitXtractor generates an XML file. The file is then distributed among working nodes via BOINC platform. Extraction of meta data significantly decreases the size of distribution and enhances privacy of the processing since no original documents are shared among working nodes.

An example of XML data extracted from the PDF document version 1.7 is shown in Figure 3.



```
  <?xml version="1.0" encoding="utf-8" ?>
- <FITCrackerFile>
  - <FITXtractor>
      <Version>0.1</Version>
    </FITXtractor>
    <Cracker>PDF</Cracker>
  - <PDF>
      <vMajor>1</vMajor>
      <vMinor>7</vMinor>
    - <Data>
        <ID1>0oIS32bHDUuXAYgc4PABAQ==</ID1>
        <ID2>LJisFcPHS0af2zEwdllLQQ==</ID2>
        <O>Bob/vEmiOX7oiMpIiuCQLsUCXbEpdw/lE=</O>
        <O_valid_salt>0FVtlpY+MZE=</O_valid_salt>
        <O_key_salt>5u1nlZAFTNQ=</O_key_salt>
        <U>0wOWU241xf4sED8TRFsiOc42bPTg4os=</U>
        <U_valid_salt>uxmqS2HoITU=</U_valid_salt>
        <U_key_salt>Nfse8+Ji10bU=</U_key_salt>
        <keyLength>256</keyLength>
        <P>4294966268</P>
        <R>5</R>
        <V>5</V>
        <MetaEncrypted>true</MetaEncrypted>
      </Data>
    </PDF>
  </FITCrackerFile>
```

Figure 3: Example of XML meta data file

The recovery process is controlled by so called *s*ecurity revision which also defines needed values extracted from the XML file. At first, security

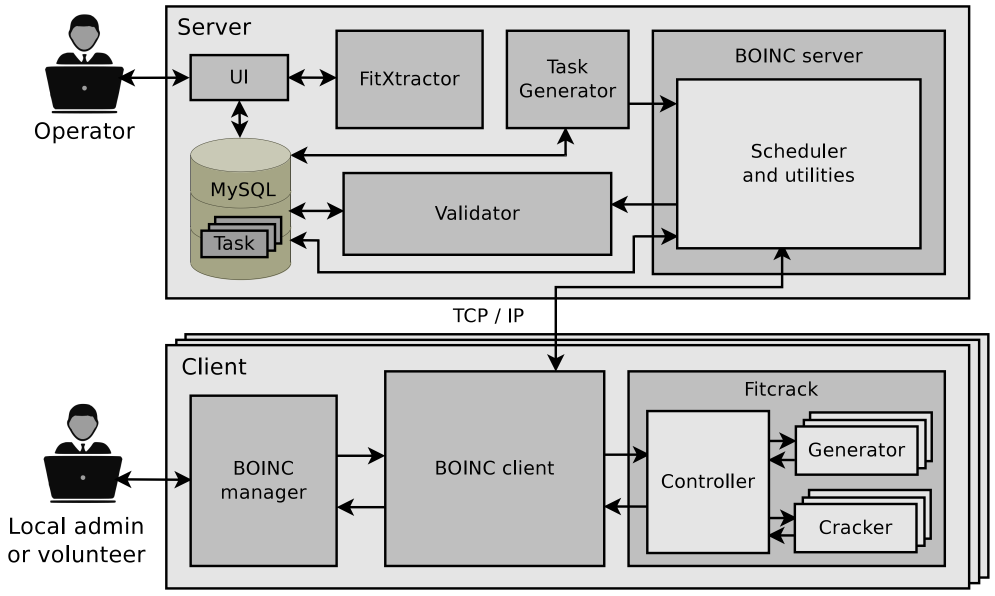


Figure 2: Distributed Password Recovery using BOINC

revision (R) is checked (it is 5 in our case). For password recovery of Rev 5 we need a specific string based on the user password (U), and the user password validation salt (U_valid_salt).

Having a generated password P, the password is concatenated with U_valid_salt and processed by SHA-256 algorithm: $P + U\_valid\_salt \overset{SHA-256}{\longrightarrow} P_{hash}$. P_{hash} is the resulting hash which is compared with a validation hash U (Adobe Systems, 2008). If these strings are equal, P is the recovered password of the given PDF file.

4.1.2 Generating Passwords

General idea of brute force password recovery is to generate all possible strings representing the password, encrypt these strings using a cryptografic algorithm (e.g., one-way hash) and compare these this string with the verification string extracted from the document.

The set of possible passwords is a *password space*. Its size depends on the maximal password length n and the alphabet Σ used. For distributed computing, ordering of the space is important. Formally, the ordered password space can be described using generator $g(i) : N \mapsto \Sigma^*$ which for given index i returns an i-th string s of password space Σ^* with maximal length n. Generator $g(i)$ can be a dictionary, a random string generator, a generator of lexicographically ordered strings, regular expressions, etc.

E.g., suppose alphabet $\Sigma = \{a, b, c, \ldots, z\}$, password length $n = 5$, and lexicographically ordered string generator. Then $g(1) = "a", g(2) = "b", g(3) = "c", g(4) = "d"$, etc.

Based on index i, we can split the entire password space into smaller parts that are distributed to working nodes. For brute force search, a lexicographical string generator is often used. In our implementation, the client contains several types of generators that are downloaded at the initialization stage, see Section 4.2.1.

Shifting the password generation to working nodes decreases the load of the server and communication between the server and working nodes. In this case, the server sends the alphabet, password length, and starting and ending index only. Working nodes use these information to select and initialize their password generator which generates an assigned password space on-the-fly and verifies each generated password with the validation string.

4.1.3 Adaptive time calculation

A critical part of distributed computing is the *task assignment*. As mentioned in Section 3.4.2, BOINC project server creates and distributes tasks to working nodes. In our architecture, each task is defined by its starting and ending index. Tasks are distributed progressively. If the password is found, other running tasks are canceled and the project finished. If the password is not found, client receives a next task represented by a new subset of the password space.

In dynamic heterogeneous environment working nodes usually have different performance based on their hardware. They can also dynamically join and leave the computing. In addition, the performance of a node can change over time. Our goal is to propose such distribution strategy that will maximize computational efficiency of the node. It means that the higher-performance nodes would receive a larger subset of password space than the lower-performance nodes.

Our approach of *adaptive time calculation* estimates how much time it would take to verify the remaining passwords on all the active nodes. Based on this time, each active node is assigned an appropriate size of password space P. The size depends on the node performance (speed). More formally, let t_p be a time (in sec) describing how much time would remaining verification take, s_i be a number of passwords (size) assigned to node i, and v_i be a current speed of node i in passwords per second.

Based on the speed, node i will be assigned $s_i = t_p \cdot v_i$ passwords. Speed v_i is determined based on the previously solved task, $v_i = \frac{s_{prev}}{t_{prev}}$.

The problem is how to choose v_i for a newly connected client. One solution is to run a *benchmark* on the client to calculate its performance.

Estimation of remaining process time t_p cannot be determined by the nodes performance only. Too low or too high values make the computing less effective:

- Lower t_p is more suitable for unstable environment where clients frequently disconnect or their performance changes. Thus, the impact of a lost task is lower. On the other hand, lower t_p increases the overhead because more communication will be needed.

- Higher t_p decreases communication overhead and nodes spend more time by the computing. In case of lost connection, recovery would be longer. Higher t_p also means less effective task distribution, namely at the end of the project. Suppose 20 clients where only 10 are active. These active nodes would be computing for another hour while others stop working since there is no more task to be assigned to them.

For efficient task distribution, we define function $proctime(t_J, s_R, k)$ that adaptively computes expected process time t_p until the entire password space is processed. Parameter t_J is an elapsed time of the computing, s_R is a number of remaining passwords to be verified and k is a number of active nodes that participate on the computing. Parameters t_J, s_R and k change over time. The function *proctime* is computed using algorithm 1. Based on remaining time t_p, each node will be assigned appropriate password space $s_i = v_i.t_p$. Thus, the remaining password space will be distributed among working nodes according to their performance. In optimal case, all nodes complete their tasks in t_p as estimated.

Algorithm 1: Adaptive calculation of t_p

```
Input: $t_J, s_R, k$
Output: $t_p$
1:  $v_{sum} = 0$
2:  forall the $client_i \in \{0, \dots, k\}$ do
3:      if $client_i$ is active then
4:          $v_i = \frac{s_{prev}}{t_{prev}}$
5:          $v_{sum} = v_{sum} + v_i$
6:      end
7:  end
8:  $t_p = \frac{s_R}{v_{sum}} \cdot \alpha$
9:  if $t_p < t_{pmin}$ then
10:     $t_p = t_{pmin}$ ;                // minimal task time
11: else if $t_J > t_{pmax}$ then
12:     $t_p = min(t_p, t_{pmax})$ ;      // maximal time
13: else
14:     $t_p = min(t_p, t_J)$ ;           // smaller tasks
15: end
16: return $t_p$
```

Lines 2 to 7 of the algorithm compute the entire speed of all active nodes. Line 8 is tricky. Normally, we would calculated t_p as $t_p = \frac{s_R}{v_{sum}}$. Here, we add parameter α called *distribution co-*

efficient that ranges from 0 to 1. In other words we say that not the entire remaining password space will be assigned, only its fraction. E.g., for $\alpha = 0.1$, only 10% of the password space P is assigned among currently active nodes. Why is not the remaining password space assigned? The answer lies in dynamic behavior of working nodes. In case that new nodes connect to the network, there would be no task for them and distributed solution would become less effective. Thus, a part of the password space is put aside hoping that more nodes will participate on the computing in the future. If not, the rest of the password space will be distributed among current nodes according the above mentioned algorithm.

Value of t_p is limited by t_{pmin} and t_{pmax}. Parameter t_{pmin} states, that the computing shorter than this value is ineffective in distributed environment, so the minimal task time is t_{pmin}. Similarly, t_{pmax} defines the maximal task time so that also slower nodes can participate in the computing. Based on our experiments, we recommend $t_{pmin} = 1$ and $t_{pmax} = 60$ minutes.

4.1.4 Task Generator

The task generator runs in loop, monitors the presence of clients and creates new tasks. In our system, two types of task are considered: (i) benchmarks and (ii) regular tasks. In case of benchmarks, the node receives a job with the same XML file as in case of the regular task. The aim of benchmark is to verify node's performance. Unlike the regular task, the node cracks as many passwords as possible during the fixed amount of time (default is 10 sec). If the benchmark fails, a new benchmark is scheduled to test the node performance.

Tasks are generated to validate passwords sequentially based on indices. Since there is no internal dependency, tasks are generated using a straightforward scheduling algorithm that assigns tasks based on the adaptive time calculation and the last password index assigned.

In case of the regular task the validator checks the result. If the password is found, the result is verified by another node and the entire project is finished, i.e., all running tasks of the project are canceled. If the result is not found, speed v_i of the current node is updated based. Results of all benchmarks and regular tasks are stored in MySQL database by the project server.

4.1.5 Security

Distributed computing is composed of a set of working nodes connected over the network. These nodes can participate fully or partially on the computing. They can also dynamically join and leave the distributed environment. Both working nodes and communication lines are considered as untrusted, hence, certain security techniques must be apply to prevent intentional or unintentional failures during the computing.

In order to provide fault-tolerance of the distributed computing, the server defines timeout T_{limit}. If a client is not able to complete the task within T_{limit} for any reason, the task is considered failed and given to another client. The failure happens whenever the client is disconnected, encounters an error, is overloaded, etc.

In untrusted environment, a client can be compromised which results in the following behavior:

a) A client reports that the password is found, but this is not true (false negative).

b) A client finds the password, but this is not reported to the server (false positive).

Case (a) can be solved by verifying the password in a *single-password mode* when another client or the server validate the result.

Case (b) is more difficult to handle since the server has no way how to check if the task (a portion of the password space) contained the password or not. The only solution is a *task replication* that is natively supported by BOINC. Replication means that each task is distributed to two or more clients and their results are compared. Unfortunately, replication significantly reduces the performance of the distributed solution.

4.2 Working Nodes

Working nodes are independent hosts connected to the project server using BOINC interface, see Figure 2. In the *initialization phase*, they download `Fitcrack` application for password recovery. Once the init phase is over, the client periodically asks the server for new tasks.

Each task includes (i) meta data of the encrypted document in XML format (see Figure 3), and (ii) a portion of the password space that should be searched for the password, see INI file in Figure 4. Depending on the password generator, a given subset of password space is generated on the fly and passwords are checked with the meta data.

```
mode:n               // regular task
charset:latin2.txt   // charset
passlength:1-8       // password from 1 to 8 chars
from:1231800416      // starting index
count:1308439710     // password space size
```

Figure 4: Example of INI file

The INI file at Figure 4 informs the working node to start a regular task of the recovery. The password generator at the node should generate passwords of length 1 up to 8 using charset *latin2*. The password space assigned to this task is bounded by a starting index and the space size.

4.2.1 Fitcrack

Fitcrack is a password recovery application developed by our team. It is based on Wrathion tool[4]. Fitcrack can run in a *multiple-password mode* for regular password recovery or in a *single-password mode* that is used for verification of a single password.

The process of password recovery is managed by the *controller* that reads an input XML file, selects a suitable password cracker and starts one of the password generators. Currently, following password generators are supported: lexicographical (brute-force) generator, user dictionary, single-layer and multi-layer masking/non-masking Markov network generator.

The *generator* is initialized by data provided in INI file. Each generator must implement $g(i)$ function (see Section 4.1.2). The generator runs either in the one-thread mode or multiple-threads mode and uses the adaptive index reservation system similar to Algorithm 1.

The *cracker* is a module responsible for password verification. For each cryptographic algorithm, a new cracker is implemented. Each cracker implements Boolean function *correct*(p) which returns `true` if the searched password matches string p, see Algorithm 2.

Algorithm 2: Password recovery process

```
Input: $i_{min}$, $i_{max}$, ; // starting and ending index
Output: plain-text password
1: forall the $i \in \{i_{min}, \dots, i_{max}\}$ do
2:     if $correct(g(i))$ then
3:         return $g(i)$
4:     end
5: end
6: return password not found
```

Indices i_{min} and i_{max} describe a portion of password space P that is searched for the password. For exhaustive search through the entire password space P, $i_{min} = 1$ and $i_{max} = |P|$.

5. RESULTS

This section shows experimental results of the distributed password recovery. The results give answers to the following research questions:

- How efficient is the scheduling algorithm?
- How high is communication overhead?
- What is the cost and energy efficiency of the solution when compared to GPU-based approach?

5.1 Testing environment

Tests were provided in heterogeneous environment of 2, 4, 8, 16, 37 and 55 CPU-nodes with the configuration described in Table 1. The table also includes configuration of GPU node with four GPU processor cards.

Nodes	Processor	Speed
1–16	Intel i5-4460, 3.2 GHz	8,000,000
17–37	Intel i3-4340, 3,6 GHz	5,000,000
38–55	Intel E8400, 3,0 GHz	2,700,000
GPU	AMD Gigabyte R9 Fury X, 4 GB	120,000,000

Table 1: Configuration of working nodes

Each line of the table describes one group of nodes involved in the computing. Line *17–37* means that nodes 1 to 16 had configuration given at line 1 and nodes 17 to 37 had configuration described in line 2. *Speed* corresponds to the benchmark tests and specifies how many passwords can be generated per second on the given processor. Thus, 15 nodes with i5-4460 CPU (line 1) have the same processing speed as one GPU processor (i.e., 15 * 8 M = 120 M).

Most of the password recovery experiments employed the worst-case test, i.e., the right password was the last password of the generated password space. In case of random passwords, it is hard to evaluate experiments since in one case a password can be found immediately because it is at the beginning of the generated password space while in the other test the search can require more time.

Our experiments provided password recovery of PDF 1.7 files, revision 5. Thus, the cracker, the password generator and the encryption algorithm were the same for all the tests.

5.2 Experiments

The first experiment shows how the number of working nodes accelerates the brute force search depending on the password length, see Figure 5.

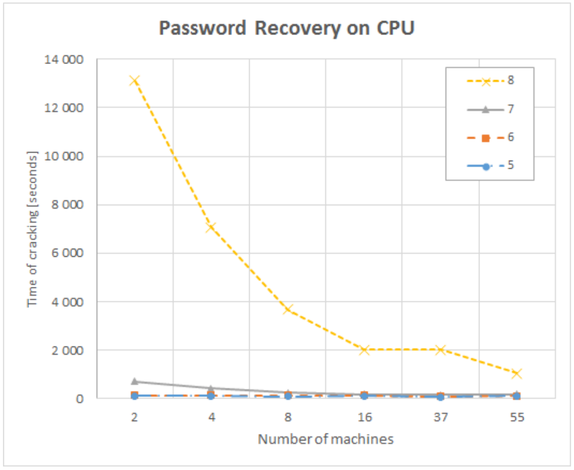


Figure 5: Distributed password recovery

The figure shows that the recovery time for passwords up to the length 7 is so fast that adding new nodes does not add any substantial acceleration to the computing. The distributed solution seems to be efficient for passwords longer than 7 characters.

This statement can be proved by password recovery of 9-character passwords, see Figure 6. In this case, the advantage of distributed password recovery is obvious. Password recovery on two or four CPU nodes is unfeasible. Eight nodes are able to find the password in 82,962 secs (cca 23 hours) while 55 nodes in 19.779 secs (5,5 hours).

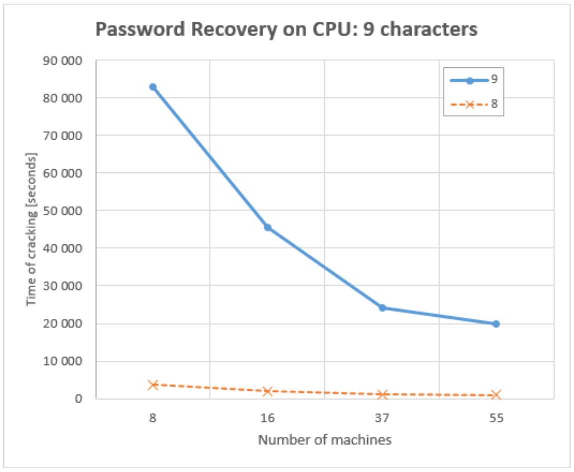


Figure 6: CPU password recovery, length 8, 9

It is interesting to compare these results with a GPU-based version. Thus, the same experiment was running on a single GPU node with one to four GPU cards active and using the same password recovery tool, see Figure 7.

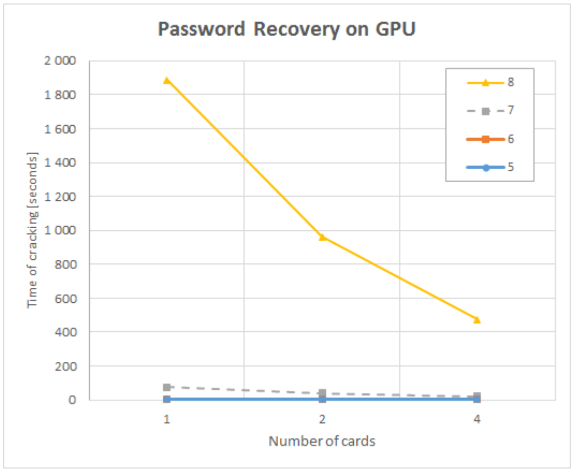


Figure 7: GPU-accelerated password recovery

Similarly to distributed solution, multiple GPU cracking is suitable for passwords longer than 7 characters. The absolute time of the recovery of 8-character password on one GPU processor corresponds to work load of 15 CPU-only nodes which proves our theoretical assumption based on benchmark tests in Table 1.

For longer passwords like nine characters, see Figure 8, the GPU-acceleration is almost linear. 9-character passwords can be solved in 49,196 secs (13,6 hours) on 1 GPU and in 12,379 secs (3,4 hours) on 4 GPUs. The graph shows comparison with 8-character password.

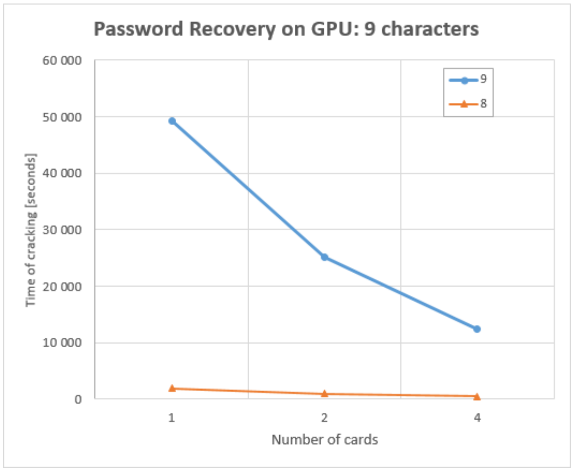


Figure 8: GPU password recovery, length 8, 9

The comparison of absolute time of distributed solution and GPU-based approach is in Table 2. The result shows where the distributed solution is comparable with a GPU version. We can see that the cluster of 16 nodes is comparable with 1 GPU cracking of 9-character password. Of course, it depends on the computational power of working nodes. However, it can be stated that distributed solution using common hardware and BOINC distribution platform can be an alternative to the GPU-based approach.

Efficiency of distributed computing and GPU-based solution is depicted in Figure 9. It represents *the percentage of the time that processors actually spend processing rather than communicating or idling* (Page & Naughton, 2005). It is computed using the following formula: $E_{ff} = \frac{\sum_{x=1}^{N} t_x}{N * T_{fin}}$ where N is the number of nodes that participated in the computing, t_x is the real time of the task processing, and T_{fin} is the final time when the password was found.

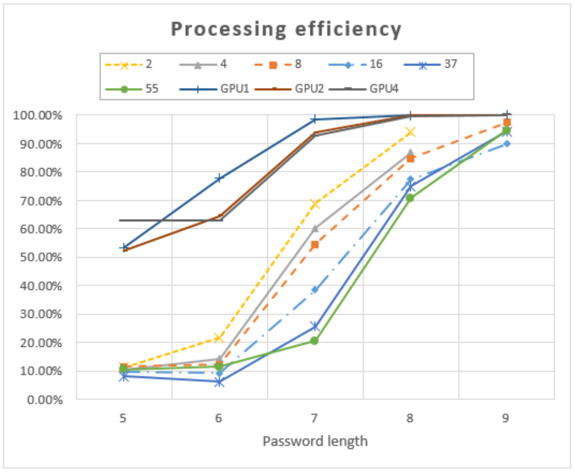


Figure 9: Efficiency of the distributed computing

The graph shows that the longer the password is, the more efficient the processing is. Efficiency of password recovery on the small password space is low because of communication overhead and benchmark tests.

For random passwords up to a given length, the result depends on where the password is found in the ordered password space. Fig. 10 shows the results of ten independent tests with random passwords up to 6 characters.

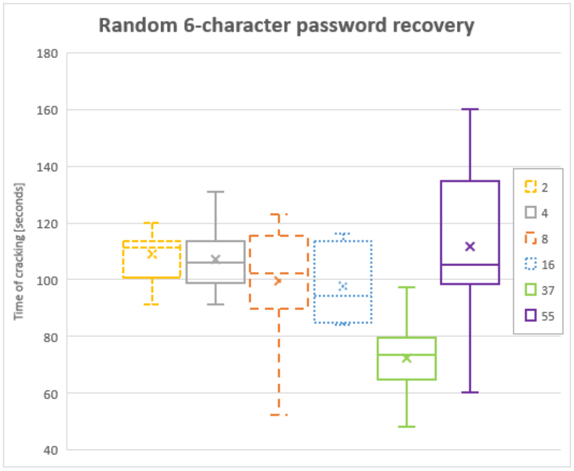


Figure 10: Random passwords up to 6 chars

We can see that the absolute time of password cracking is similar for 2 to 55 nodes. Since the cracking is so fast, the total time is influenced more by the overhead.

Password length	Number of nodes								
	2	4	8	16	27	55	1 GPU	2 GPU	4 GPU
5	100	101	93	104	60	100	2,79	2,87	4,02
6	130	132	132	134	87	99	5,8	3,87	4,01
7	714	415	236	167	186	184	74	39,07	21,21
8	13 125	7 051	3 679	2 025	1 177	1045	1 885	961	475
9	–	–	82 962	45 567	24 011	19 779	49 196	25 120	12 379

Table 2: Time (in secs) of distributed and GPU-based password recovery

Real acceleration of distributed solution is more visible for longer passwords. Figure 11 shows the test for the 8-character password. The

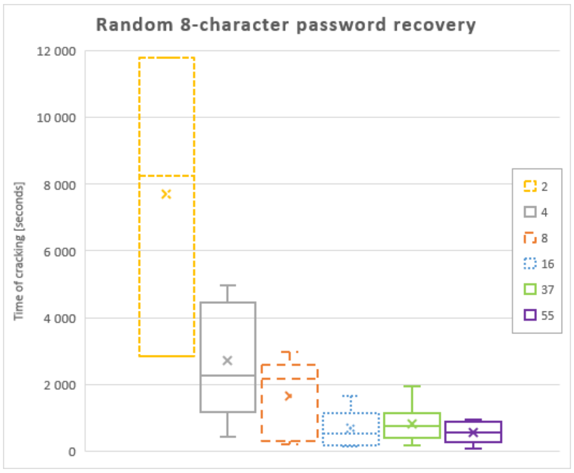


Figure 11: Random passwords up to 8 chars

graph shows substantial acceleration for clusters of 16 and more nodes.

5.3 The Cost Model

An important question of forensic experts is what is the cost of the password recovery? In this section, we present the comparison of cost models for distributed solution and GPU-based approach. For our cost model, workstations of Table 1, line 1, are considered.

Table 3 gives an overview of the cost of working nodes based on 2015 prices. The *unit price* is a price of processing 1,000,000 passwords per second on the given hardware. The unit price is constant for distributed computing on nodes with the same configuration. In our case, it is \$79. The unit price of GPU solution is cheaper when inserting additional GPU cards. The price includes not only GPU processors, but also motherboard, PCIe slots, additional CPU cooler, etc.

Node(s)	Speed (pwd / sec)	Price (USD)	Price / Unit USD
1	8,000,000	639	79
2	16,000,000	1278	79
4	32,000,000	2556	79
8	64,000,000	5 112	79
16	128,000,000	10 224	79
1 GPU	120,353,069	2 337	19
2 GPU	240,706,138	3 151	13
4 GPU	481,412,276	4 440	9

Table 3: Relative cost of the solution

The table shows that GPU-based password recovery is cheaper solution than building a new CPU cluster. If the cluster is already available, then the distributed solution can provide a feasible alternative to the high-performance multi-GPU node with high initial costs.

Another important factor connected with the cost model is the energy consumption, see Table 4. The first two columns show the power consumption (in Watts) of working nodes during the idle state (no password recovery) and during the processing (load state). Next two columns represent the real costs of cracking the passwords of lengths 8 and 9 with unit price of \$0.12 per kWh.

The table also shows that the power consumption of the 16-nodes cluster is similar to a 4-GPU node. However, the price of consumed energy is lower for GPU because of faster computing.

The overall power consumption (in watt-hours) is calculated as the electrical power consumed during the effective computing (P_load) and the power consumed during the idle time (communication, etc.): $P_{total} = (N * T_{fin} * (E_{ff} * P_{Load} + (1 - E_{ff}) * P_{Idle}))/3600$, where N is the number of nodes, T_{fin} is the time when the pass-

Node(s)	Idle (W)	Load (W)	Price L8 (USD)	Price L9 (USD)
1	31	58	-	-
2	62	116	0.0493	-
4	124	232	0.0518	-
8	248	464	0.0531	1.2678
16	496	928	0.0561	1.3437
1 GPU	182	370	0.0232	0.6067
2 GPU	182	541	0.0173	0.4530
4 GPU	182	856	0.0135	0.3532

Table 4: Power consumption and the cost

word was found, E_{ff} is the efficiency, P_{Load} or P_{Idle} is the power consumption during the password recovery or the idle state, respectively.

The total power consumption spent on cracking password of length 5 to 8 on CPU and GPU nodes is shown in Figure 12.

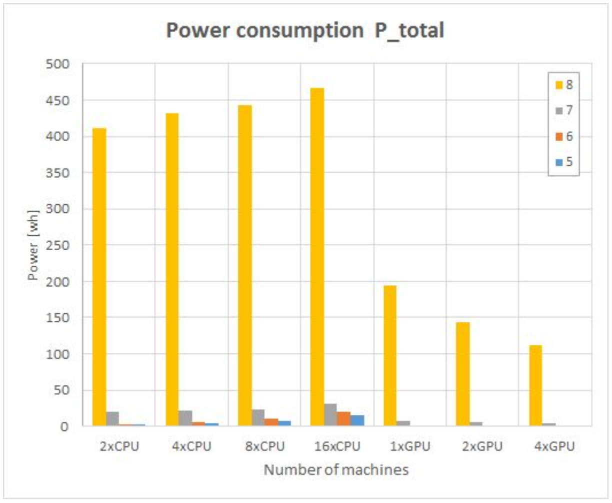


Figure 12: Power consumption of the computing

The password recovery is generally more power efficient on GPU cards in comparison to CPU units. When adding new nodes, the total power consumption P_{total} softly rises on CPU nodes while it goes down with adding new GPU cards to the GPU node for the computing.

6. CONCLUSION

Password recovery of encrypted documents is an important issue for digital forensic experts. Today, many solutions focus on GPU-based acceleration. In this paper, we presented a distributed alternative built upon a just-available hardware with similar performance for shorter passwords. If no workstations are available, the GPU-based solution is more efficient.

We also discussed issues connected with distributed environment. First, individual tasks should be distributed based on the real performance of working nodes. Uniform task distribution to nodes with variable speed can degrade the real performance of the cluster. Second, distributed solution supports is easily scalable and can employ workstations outside office hours. Platforms like BOINC provide an easy deployment of the distributed computation.

6.1 Future Work

For the future, we plan to explore distributed password recovery in heterogeneous environment that combines CPU and GPU nodes. Besides the issue of hardware dependence, task distribution algorithm must be revised because the performance benchmark of a GPU node is incomparable to a CPU node which could result in unbalanced task distribution.

In this work we were focused on encrypted documents and archives only. However, the presented approach and environment can be extended to encrypted file systems using TrueCrypt, Bitlocker, PGP, File Vault, or SafeGuard.

Supposing TrueCrypt, this tool creates a virtual encrypted disk in the form of a single file, it can encrypts a disk partition, or the entire disk storage. For all these cases, FitXtractor can be extended to extract the partition encrypted headers and create XML meta data file for Fitcrack. Like to other encrypted documents, the probability of success depends on the strength of the password and the password generator. Unfortunately, TrueCrypt supports eight different encryption algorithms and three hash algorithms which makes $8*3 = 24$ combinations. Since there is no way how to identify the algorithms used, all combinations should be tested which complicates the recovery (Zhang, Zhou, & Fan, 2014).

The password discovery of BitLocker is also feasible (Dija, Balan, Anoop, & Ramani, 2011). Since BitLocked drive is loaded at the boot time, it is possible to perform a memory dump and search for a `.BEK` file containing the encryption key. Another approach is the usage of Bootkits to compromise the kernel (Yi-ming & Sheng-li, 2010).

ACKNOWLEDGMENTS

This work was supported by The Ministry of Education, Youth and Sports from the National Program of Sustainability (NPU II) project "IT4Innovations excellence in science", LQ1602 and by project "Research and application of advanced methods in ICT", FIT-S-14-2299.

REFERENCES

Adobe Systems. (2008, June). *Adobe Supplement to the ISO 32000. BaseVersion: 1.7. Extension-Level:3* (Tech. Rep.).

Anderson, D. P. (2004, Nov). BOINC: a system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop Grid Computing* (pp. 4–10).

Apostal, D., Foerster, K., Chatterjee, A., & Desell, T. (2012, Dec). Password recovery using MPI and CUDA. In *Proc. of HiPS, 2012* (p. 1-9).

Dell'Amico, M., Michiardi, P., & Roudier, Y. (2009). Measuring Password Strength: An Empirical Analysis. *CoRR*, *abs/0907.3402*.

Dija, S., Balan, C., Anoop, V., & Ramani, B. (2011). Towards Successful Forensic Recovery of Bit-Locked Volumes. In *6th Conference on System of Systems Engineering (SoSE)* (pp. 317–322).

Hranický, R., Matoušek, P., Ryšavý, O., & Veselý, V. (2016). Experimental Evaluation of Password Recovery in Encrypted Documents. In *Proceedings of ICISSP 2016* (pp. 299–306). Roma.

Kang, S. J., Lee, S. Y., & Lee, K. M. (2015, Aug). Performance Comparison of OpenMP, MPI, and MapReduce in Practical Problems. *Advances in Multimedia*, 9.

Kelley, P., Komanduri, S., Mazurek, M., Shay, R., Vidas, T., Bauer, L., ... Lopez, J. (2012, May). Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms. In *IEEE Symposium on Security and Privacy* (p. 523-537).

Marechal, S. (2007). Advances in password cracking. *Journal in Computer Virology*, *4*(1), 73–81.

Marks, M., & Niewiadomska-Szynkiewicz, E. (2014). Hybrid CPU/GPU Platform For High Performance Computing. In *Proc. of the 28th ECMS.*

Page, A. J., & Naughton, T. J. (2005). Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. In *19th int. parallel and distr. processing* (pp. 189a–189a).

Pellicer, S., Pan, Y., & Guo, M. (2004). Proc. of Grid and Cooperative Computing. In H. Jin, Y. Pan, N. Xiao, & J. Sun (Eds.), (pp. 679–686). Berlin, Heidelberg: Springer.

Reimann, S. (2013). *Analyzing the Structure of Passwords to Improve Strength Measurement and Password Cracking* (MSc. Thesis). Ruhr-Universität, Bochum, DE.

Thing, V. L., & Ying, H.-M. (2009). A novel time-memory trade-off method for password recovery. *Digital Investigation*, *6, Supplement*(0), S114 - S120.

Ur, B., Kelley, P. G., Komanduri, S., Lee, J., Maass, M., Mazurek, M. L., ... Cranor, L. F. (2012). How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation. In *Usenix security* (pp. 65–80).

Yi-ming, J., & Sheng-li, L. (2010). The Analysis of Security Weakness in BitLocker Technology. In *Proc. of the Int. Conference on Networks Security, Wireless Communications and Trusted Computing* (pp. 494–497).

Zhang, L., Zhou, Y., & Fan, J. (2014). The Forensic Analysis of Encrypted Truecrypt Volumes. In *Int. Conference on Progress in Informatics and Computing (PIC)* (pp. 405–409).