Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2016) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC-2016)

Taming Energy Cost of Disk Encryption Software on Data-Intensive Mobile Devices

Yang Hu[a], John C.S. Lui[b,*], Wenjun Hu[c], Xiaobo Ma[a], Jianfeng Li[a], Xiao Liang[a]

[a] *Xi'an Jiaotong University, No.28, Xianning West Road, Xi'an, Shannxi, 710049, China*
[b] *The Chinese University of Hong Kong, Shatin, NT, Hong Kong, China*
[c] *Palo Alto Networks Inc., Singapore*

Abstract

Disk encryption is frequently used to secure confidential data on mobile devices. However, the high energy cost of disk encryption poses a heavy burden on those devices with limited battery capacity especially when a large amount of data needs to be protected by disk encryption. To address the challenge, we develop a new kernel-level disk encryption software, *Populus*. Almost 98% of *Populus's* encryption/decryption computation is not related with the input plaintext/ciphertext, so we accomplish the computation in advance during initialization when a consistent power supply is available. We conduct cryptanalysis on *Populus* and finally conclude that state-of-the-art cryptanalysis techniques fail to break *Populus* in reasonable computational complexity. We also conduct energy consumption experiments on *Populus* and dm-crypt, a famous disk encryption software for Android and Linux mobile devices. The experimental results demonstrate that *Populus* consumes 50%-70% less energy than dm-crypt.

Keywords: privacy protection; disk encryption; energy-efficient computing.

1. Introduction

In recent years, mobile devices, such as smartphones, smartwatches and mobile video surveillance devices[1], have become an integral part in our daily life. Meanwhile, mobile devices are usually facing profound security challenges, especially when being *physically* controlled by attackers. For example, due to device loss or theft, data leakage in mobile devices happens more frequently than before[2]. To deal with the aforementioned security challenge, mobile devices can encrypt secret data and store its ciphertext locally on itself, which is also known as *disk encryption*[3]. This method attracts extensive attention in industry and academia[4]. Generally speaking, there are two types of disk encryption solutions: software and hardware solutions. This paper mainly focuses on software solutions as they usually have advantages in compatibility and scalability.

* Corresponding author. Tel.: (852) 3943-8407 ; fax: (852) 2603-5024.
E-mail address: cslui@cse.cuhk.edu.hk

However, for data-intensive applications such as mobile video surveillance[1] and seismic monitor[5], the whole energy consumption of mobile devices highly rises after applying existing disk encryption software. One evidence proposed by Li et al. states that for data-intensive applications nearly 1.1-5.9 times more energy is required on commonly-used mobile devices when turning on their disk encryption software[6]. Worse, mobile devices are usually battery-powered in order to improve portability. For example, sometimes mobile video surveillance device is equipped on a multi-rotor unmanned aerial vehicle, so battery becomes its sole power supply[7]. Due to mobile devices' limited battery capacity, existing disk encryption software may strongly affect their normal usage.

The significant energy overhead of existing disk encryption software can be explained by the following two reasons. First, the ciphers used in existing disk encryption software contain many CPU and RAM operations, which are usually not energy-efficient. Second, massive data needs to be protected by disk encryption for data-intensive application, which multiplies its energy consumption. For instance, mobile video surveillance devices need to real-timely record and securely store a large amount of video data[7]. According to our experiments, nearly 1/3 of energy consumption comes from existing disk encryption software in mobile video surveillance.

In fact, energy consumed by CPU and RAM operations tends to become more prominent than other conventional concerns such as screen and network communication, especially when disk encryption software participates in data-intensive tasks. About *six years ago*, about 45%-76% of daily energy consumption came from screen and GSM when disk encryption software is disabled[8]. However, the distribution of energy consumption has been changed dramatically in recent years due to software&hardware optimization and usage habit transformation. A recent study[9] shows that for typical usage only about 28% of energy consumption results from screen and GSM, while CPU and RAM spend about 35% energy and become the largest energy consumption source in mobile devices when disk encryption is disabled. In addition, both[8] and[9] measures energy consumption without enabling disk encryption function. So when considering that existing disk encryption software owns many CPU and RAM operations, we believe that the energy consumption percentage of CPU and RAM may be more higher than 35% if data-intensive mobile devices enable disk encryption software. Li's experiment results[6] exactly verified it. Hence, to build an energy-efficient mobile system, reducing the energy consumption in disk encryption software is a rational starting point.

To reduce energy consumption of disk encryption software, some researchers try to reduce the number of CPU or RAM operations in disk encryption software. But it is really challenging to make disk encryption software both energy-saving and cryptographically secure in this way. Generally speaking, the less computation disk encryption software needs, the less energy it costs, but possibly the more insecure in cryptography. For example, some trials[10] are faced with challenges in terms of cryptography[11]. In existing *cryptographically secure* disk encryption software, the disk encryption software used in Linux and Android, also known as *dm-crypt*, theoretically owns less computation than others. But our experimental results show that nearly 30%-50% of mobile device's energy consumption comes from dm-crypt for typical usage of data collection and transmission. So the energy consumption of state-of-the-art disk encryption software is still unnegligible.

In this paper, we design and implement a new energy-efficient kernel-level disk encryption software, *Populus*. The basic idea behind *Populus* is to extract the "*input-free*" computation from the cipher in disk encryption software and accomplish it during initialization, where the input-free computation refers to the cipher's operations that are not involved with the input text (i.e., plaintext or ciphertext). For example, in AES-CBC cipher, its key expansion can be regarded as input-free computation. The initialization's energy consumption is not considered in this paper because it is performed only once when a mobile device is first used and a consistent power supply is usually available. Therefore, the more input-free computation we can extract, the more energy we can save.

However, the ciphers used in existing disk encryption software only have a little input-free computation. For example, we found that input-free computation of AES-CBC cipher accounts for at most 1% of all its computation. To improve the proportion of input-free computation, *Populus* first generates *pseudo random numbers* (*PRNs*) and global matrices in an input-free manner. Next, it use those PRNs and global matrices to construct temporary matrices and then conduct carefully designed matrix multiplication when encrypting user's privacy. Each PRN can only participate in disk encryption once so that sufficient PRNs are usually needed for data-intensive application. To protect those PRNs and matrices, *Populus* encrypts them in an *iterative* manner. In this way, *Populus* can save much energy because almost 98% of its computation is input-free and the residual real-time computation is much smaller than current disk encryption software. In addition, *Populus* costs acceptable extra storage space (typically $\leq$ 256MB) for those PRNs and matrices.

To assess *Populus* in the respect of cryptographic security and energy efficiency, we conduct cryptanalysis on *Populus* and a series of energy consumption experiments on both *Populus* and dm-crypt. Finally we find that *Populus* can defend against state-of-the-art cryptanalysis techniques and simultaneously consume less energy than dm-crypt. Our contribution can be generalized into the following items.

- To the best of our knowledge, this paper is the first work focusing on extracting input-free computation from disk encryption software, which can be used to reduce its energy consumption.
- We design and implement an energy-saving kernel-level disk encryption software *Populus* that can both defend against state-of-the-art cryptanalysis techniques and save 50%-70% more energy than dm-crypt.

The remainder of the paper is organized as follows. Section 2 explains why existing disk encryption software is lack of input-free computation and how to improve its proportion in *Populus*. Section 3 presents our system *Populus* in detail. Section 4 evaluates the cryptographic security of *Populus*. Section 5 presents the experimental results of mobile devices' energy consumption. Section 6 summarizes related work. Concluding remarks then follow.

2. Input-Free Computation

In this section, we present more details about input-free computation. We first introduce the design consideration of existing disk encryption software and explain why they are lack of input-free computation. Then, we show the basic idea of *Populus* and illustrate why it can improve the proportion of input-free computation.

Existing disk encryption software including dm-crypt is usually based on *tweakable scheme*[12], where each disk sector should correspond to an independent key used only for its encryption and decryption. However, in practice, a user only provides one master key. To solve this problem, most of existing disk encryption software achieves tweakable scheme in the following two steps: **1)** produce sector-specific keys based on master key and sector ID; **2)** use sector-specific key to encrypt certain disk sector with a block cipher.

Due to the fact that attackers can get multiple (plaintext, ciphertext) pairs in the same disk sector, they can conduct *chosen-plaintext attack* (*CPA*) by exploiting (plaintext, ciphertext) pairs sharing same sector-specific key. Hence, to secure the whole crypto system, the block cipher in **2)** must have the ability to defend against CPA. To achieve this, one effective solution is to construct a block cipher in *substitution-permutation network* (*SPN*)[13]. Unfortunately, we find that nearly all SPN-based block ciphers have a little input-free computation because their core components, *substitution box* and *permutation box*, directly or indirectly rely on input.

To improve input-free computation, we give up aforementioned tweakable scheme and SPN when designing *Populus*. Instead, we construct *Populus* based on *nonce-based scheme*[14]. *Populus's* core design can be briefly described as follows: **a)** for i^{th} encryption, produce an independent temporary key based on master key and *i*; **b)** use the temporary key and a light-weight block cipher to accomplish i^{th} encryption. Our design has four advantages. First, it is compatible to tweakable scheme. Second, it makes attackers hard to get multiple (plaintext,ciphertext) pairs sharing same key, and thereafter basically eliminate the threat from CPA. Third, nearly all procedures in **a)** are input-free. Forth, SPN becomes unnecessary in **b)**, which gives us more freedom to design a light-weight block cipher owning much input-free computation. As a trade-off, our scheme needs extra storage space for input-free computation. Fortunately, the storage space can be reduced to an acceptable level by carefully designing the temporary key production method in **a)** and the block cipher in **b)**. In Section 3, we complement details regarding the design and implementation of *Populus*.

3. Populus: An Energy-Saving Disk Encryption Software System

Populus consists of two parts: *system initialization* and *real-time encryption/decryption*. We perform system initialization once when a mobile device is first used and we assume that a consistent power supply is available and the energy consumption is not a concern during system initialization. *Populus* initially accomplishes all input-free computation and stores its result on disk, which is used for processing real-time encryption and decryption requests later. *Populus* works at a *512-byte disk sector* level, and it allows users to manually configure *private disk sectors*, which store users' confidential information. For each private disk sector, *Populus* initially assigns it a *temporary key*, which is used for encrypting/decrypting the confidential data on it. Each temporary key can only be used for one encryption. If a sector has 'consumed' its temporary key due to encryption, *Populus* will recycle its current temporary key and allocate a new temporary key for its next encryption. We design *Populus* for 64-bit systems because 64-bit processors are popular for mobile devices. Throughout the paper, the default value of a number's size is 64 bits unless stated otherwise. Next, we introduce each part of *Populus* in detail.

3.1. System Initialization

The system initialization includes three input-free modules: *PRN generator*, *master key generator* and *IFCR encryption*. Here, *IFCR* is the abbreviation of *input-free computation result*. PRN generator produces PRNs, which are basic for generating master key and real-time encryption/decryption. Next, *Populus* encrypts IFCR and then stores it on disk.

3.1.1. PRN Generator

To produce PRNs, we use Salsa20/12 stream cipher, which has been extensively studied and found to produce PRNs of very high quality[15]. Salsa20/12 requires a 320-bit input, hence we use the SHA3 algorithm[16] to map a user's arbitrary-length key into a 384-bit number, and extract the first 320-bit *hash key* as the input of Salsa20/12 stream cipher. PRNs are mainly used for master key production and real-time encryption/decryption, which are separately named *MK-PRNs* and *RT-PRNs*.

3.1.2. Master Key Generator

Populus generates master key using MK-PRNs. We define a square matrix A is 2^{64} *modular invertible* when there exists a matrix B such that $AB = I \mod 2^{64}$, where I is the identity matrix. If this is the case, then the matrix B is uniquely determined by A and is called the modular inverse of A (mod 2^{64}). For simplicity, we denote it by A^{-1} in this paper. We denote master key as $U = (U^{(1)}, \ldots, U^{(125)})$, where each $U^{(i)} = \begin{pmatrix} u^{(i)}_{1,1} & u^{(i)}_{1,2} \\ u^{(i)}_{2,1} & u^{(i)}_{2,2} \end{pmatrix}$, $1 \leq i \leq 125$, is a 2×2 matrix and $U^{(i)}$ is *modular invertible*, which is critical for the real-time encryption and decryption discussed later.

We randomly select matrices $U^{(1)}, \ldots, U^{(125)}$ from the set of *modular involutory* matrices based on the Acharya's method[17]. Here a modular involutory matrix is defined as a matrix whose modular invertible matrix is itself. Since there exists 7.66×10^{38} modular involutory matrices[18], the number of all possible U is $(7.66 \times 10^{38})^{125} \approx 3.38 \times 10^{4860}$, which is much larger than the size of our hash key space (i.e., $2^{320} \approx 2.14 \times 10^{96}$). Therefore, the master key is more difficult to brutally crack than the hash key.

3.1.3. IFCR Encryption and Decryption

To protect IFCR (i.e., RT-PRNs and master key), *Populus* encrypts them and then stores them on disk. During real-time encryption/decryption, *Populus* decrypts master key and RT-PRNs from disk. Later, we will introduce more detail in Section 3.3.

3.2. Real-Time Encryption and Decryption

Populus performs disk encryption/decryption when the file system writes/reads data on disk in real time. We introduce each of its modules as follows.

3.2.1. Transparent Encryption and Decryption

Our transparent encryption and decryption is based on matrix multiplication in *modular linear algebra*[19]. In cryptography, matrix multiplication has achieved Shannon's diffusion and it dissipates statistical structure of the plaintext into long-range statistics of the ciphertext to thwart cryptanalysis based on statistical analysis[13]. However, matrix multiplication is usually computationally intensive. For example, a general matrix multiplication between a 64×64 matrix and a 64×1 matrix requires $64 \times 64 + 64 \times 63 + 128 = 8256$ operations.

To reduce its computation, *Populus* only constructs 125 64×64 sparse matrices $H^{(i)} = \begin{pmatrix} I_{62-|63-i|} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & M^{(i)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_{|63-i|} \end{pmatrix}$ where $i \in \{1, \ldots, 125\}$, I_i is the i-dimensional identity matrix, and $M^{(i)}$ is a 2×2 modular invertible matrix. Then *Populus* computes $H^{(125)} \ldots H^{(1)}P$ as encryption or $(H^{(1)})^{-1} \ldots (H^{(125)})^{-1}C$ as decryption where P is a 64×1 matrix as one 512-byte plaintext and C is a 64×1 matrix as one 512-byte ciphertext. Exploiting $H^{(i)}$ is a sparse matrix, 125 64-dimensional matrix multiplications can be simplified to 125 2-dimensional matrix multiplications. The simplified encryption and decryption only consists of $125 \times (2 \times 2 + 2 \times 1) + 128 = 868$ operations.

3.2.2. *Temporary Key Manager*

Each temporary key consists of $125 \times 2 \times 2 \times 8 = 4000$ bytes, therefore storing all temporary keys requires a large storage space. To solve this problem, *Populus* computes its M based on U, R_{2j-1} and R_{2j} for j^{th} encryption, where $R = (R_1, ..., R_d)$ denote RT-PRNs, $R_i, 1 \leq i \leq d$, is a pseudo random number, and d is the size of R. Note that U is shared by all temporary keys' construction and its size is 4000 bytes, so on average, we only need about 16 bytes for storing a temporary key. Then, for j^{th} encryption, we compute each $m^{(i)}_{p,q}$ $(p, q = \{1, 2\})$ in $M^{(i)}$, as

$$m^{(i)}_{p,q} = \begin{cases} [2(u^{(i)}_{p,q} \oplus R_{2j-1}) + [u^{(i)}_{p,q}]_2]_{2^{64}}, & i = 1, \\ [2(u^{(i)}_{p,q} \oplus R_{2j-1} \oplus R_{2j}) + [u^{(i)}_{p,q}]_2]_{2^{64}}, & i = 63, \\ [2(u^{(i)}_{p,q} \oplus R_{2j}) + [u^{(i)}_{p,q}]_2]_{2^{64}}, & i = 125, \\ u^{(i)}_{p,q}, & \text{otherwise.} \end{cases} \tag{1}$$

Here, we use the notation $[m]_n$ to denote the function $m \mod n$, i.e., $[m]_n = m \mod n$. We prove $M^{(i)}$ is modular invertible in[20].

Considering that RT-PRNs can only be used once, d should be as large as possible in order to securely store mass data. But if d is too large, RT-PRNs will occupy a lot of storage space so that there may be no enough space for user's data. To mitigate this contradiction, *Populus* only stores a balanced amount of RT-PRNs that can support real-time encryption/decryption before battery uses up and then replenishes RT-PRNs during device charging or battery replacement.

After applying our method, only small storage space of RT-PRNs is able to satisfy most of applications in practice. Suppose that on average mobile devices require to securely store l-byte data each day and can work t days without enabling disk encryption. *Populus* needs at most $d = lt/256$ pseudo random numbers in RT-PRNs. For example, as for smartphone, we let $t = 4$ and $l = 2^{31}$ so that only 256 MB are required to store $d = 2^{25}$ pseudo random numbers.

3.3. *Iterative Encryption and Decryption on IFCR*

In Section 3.1.3, we have briefly introduced the function of IFCR encryption and decryption. However, IFCR decryption may cost much energy if we choose existing block ciphers as its encryption/decryption algorithm. For example, if *Populus* uses AES-CBC to encrypt IFCR, nearly all encrypted IFCR should be decrypted for each time of transparent encryption, which obviously costs lots of energy.

To reduce the aforementioned energy cost, we propose a dedicated encryption method called *iterative encryption* for IFCR protection. The basis idea of iterative encryption comes from our observation that *Populus* only needs one new master key (4000 bytes) and k new RT-PRNs ($16k$ bytes) when encrypting k disk sectors (512 bytes for each disk sector) whose data is never changed. Considering that master key and RT-PRNs are never changed once generated, we iteratively encrypt them as follows: **a)** If IFCR only occupies $k \leq 9$ disk sectors in all, *Populus* directly encrypts them through a SPN-based cipher (e.g., AES-CBC); **b)** If IFCR occupies $k > 9$ disk sectors, *Populus* produces another new IFCR including one new master key and $\lceil (16k + 4000)/512 \rceil$ new RT-PRNs and use them to encrypt original IFCR through our proposed transparent encryption method; **c)** Encrypt new IFCR by repeating **a)** and **b)**. As for *iteration decryption*, just reverse the whole process of iteration encryption.

We can use *master method* to prove that the computation complexity of our iterative encryption/decryption is $O(log(n))$. Here, n denotes the number of disk sectors occupied by IFCR. Compared with AES-CBC which needs $O(n)$ computation in same task, our iterative encryption/decryption saves much energy.

4. Security Analysis

To rigorously assess *Populus's* security, we use state-of-the-art cryptanalysis techniques such as *linear*, *differential*, *algebraic*, *slide*, and *Biclique attacks* on *Populus*. Finally, we show that all of them fail to break *Populus* in reasonable computational complexity. The detail of our analysis process can be found in[20].

5. Energy Consumption Evaluation

In this section, we use *Monsoon power monitor*[21] to measure energy consumption of the whole mobile device and estimate the energy cost by disk encryption software. We choose Google Nexus 4 smartphone with Android 5.0 OS as our tested mobile device. To compared with Populus proposed in this paper, dm-crypt is chosen as the baseline for the following two reasons. First, the architecture of dm-crypt is similar to other popular disk encryption software and their computation is close. So we can use dm-crypt as a representative of existing disk encryption software. Second, dm-crypt is compatible with Android. So it is convenient for us to conduct energy consumption experiments on our Google Nexus 4 smartphone.

5.1. Evaluation on Typical Usages for Mobile Device

We conduct a series of experiments to measure the energy consumption of mobile device's typical usage. Through those experiments, we can verify whether enabling dm-crypt tremendously raises the whole device's energy consumption and whether *Populus* can mitigate it.

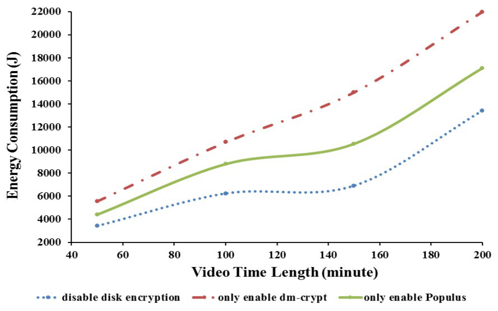
We also design three configurations for the mobile device: only enabling dm-crpyt, only enabling *Populus* and disabling any disk encryption. For each configuration, we measure the mobile device's whole energy consumption in four typical usage: video recording, video playing, data sending throgh WIFI, data receiving through WIFI. As for video playing and recording, video format is *mp4*, video resolution is 480×270, the choices of video length are 50min, 100min, 150min and 200min and video quality is of high definition. As for WIFI network, the choices of transferred data size are 256MB, 512MB, 768MB,. . ., 2048MB.

Then we introduce our experiments separately. Video playing is a common function for handheld mobile device such as smartphone and its energy consumption status is shown in Fig. 1(a). Note that when playing an encrypted video, decryption is necessary so that part of energy consumption comes from *Populus* or dm-crypt if they are enabled. As you can see, nearly 1/2 of energy is cost by dm-crypt and *Populus* can reduce it to nearly 1/4.

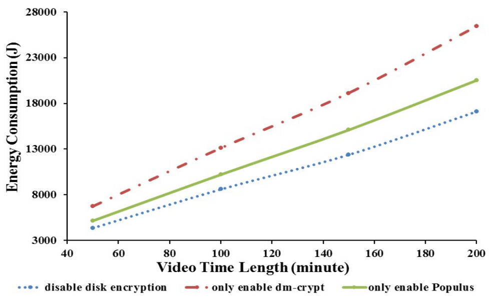
We also present relevant experimental results of video recording in Fig. 1(b), as video recording on mobile device is widely used in personal life, industry and military (e.g., mobile video surveillance[1]). Obviously when recording a secret video, disk encryption is necessary so that part of energy consumption comes from *Populus* or dm-crypt if they are enabled. Our experimental results show that nearly 1/3 of energy is cost by dm-crypt and *Populus* can reduce it to nearly 1/6.

As for network data transference, Fig. 1(c) demonstrates mobile device's energy consumption when it sends data to remote terminal through WIFI network. Here, data has been encrypted by disk encryption software in advance so that data decryption before network transference should be considered if disk encryption software is enabled. Apparently, there is an approximate linear relation between transferred data size and mobile device's energy consumption. On average, 51% of energy consumption on mobile device is cost by dm-crypt and *Populus* can reduce it to 20%.

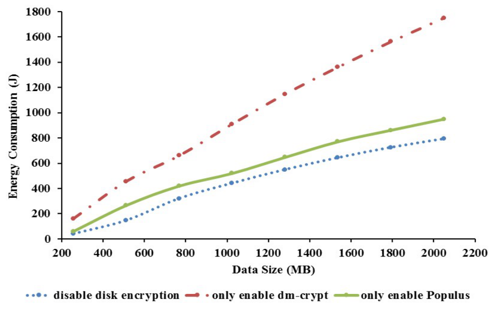
Fig. 1(d) shows mobile device's energy consumption when it receives data from remote terminal through WIFI network. Here, we regulate that those received data will be encrypted by disk encryption software if enabled. As you can see, it is not a pure linear relation between data size and mobile device's energy consumption. In detail, the energy consumption of the mobile device enabling dm-crypt is close to the mobile device enabling *Populus* when data size is small and gradually changed to linear relation as data size grows larger. Due to file system buffer and disk I/O buffer, part of received data may be lazily cached in buffer so that disk encryption may not be fully triggered. On average, 56% of energy consumption is cost by dm-crypt and *Populus* can reduce it to 25%.



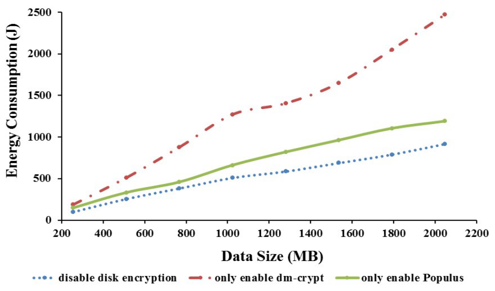
(a) Video Playing



(b) Video Recording



(c) Data Sending



(d) Data Receiving

Fig. 1: Energy consumption on playing&recording and data sending&receiving

5.2. *Evaluation on Pure Disk Encryption/Decryption Operations*

To compare dm-crypt with *Populus*, one effective way is to compute the energy consumption of pure disk encryption operations in dm-crypt and *Populus* and then compute the improvement percentage. However, both of them can not be directly measured by Monsoon power monitor. To solve this problem, we design a comparison model to estimate them.

Next, we formally introduce our comparison model. The energy cost of dm-crypt is denoted by AE_i and the energy cost of *Populus* is denoted by PE_i. Here, i denotes the file size in certain experiment. For example, when recording a video, i denotes the video file size. Let $GE_i = \frac{AE_i - PE_i}{AE_i}$ denote the percentage of energy that *Populus* saves in comparison with dm-crypt when processing i-megabyte file, and we use $\overline{GE}$, the average of all GE_i, to compare the energy consumption between *Populus* and dm-crypt. We regulate three different configurations as: $Conf.1$, all disk encryption systems are disabled; $Conf.2$, only dm-crypt is enabled; $Conf.3$, only *Populus* is enabled.

We first measure the energy consumption $EC_{i,j}$ $(j \in \{1, \ldots, 3\})$ and the time cost $ET_{i,j}$ $(j \in \{1, \ldots, 3\})$ of our mobile phone with different $conf.j$ $(j \in \{1, \ldots, 3\})$. In addition, we observe that the energy consumption of Android OS is stable, so we denote SP as the energy cost of system per second, and SP can be directly computed by measuring the power consumption when our mobile device is idle. Then we compute GE_i based on $EC_{i,j}, j \in \{1, \ldots, 3\}$, $ET_{i,j}, j \in \{1, \ldots, 3\}$, and SP. Let FE_i denote the energy consumption of the pure file and disk operations on i-byte file excluding disk encryption/decryption and $SE_{i,j}, j \in \{1, \ldots, 3\}$ denote the energy consumption of Android OS for $Conf.j$. Considering $SE_{i,j} = SP \cdot ET_{i,j}$,$EC_{i,1} = FE_i + SE_{i,1}$, $EC_{i,2} = FE_i + SE_{i,2} + AE_i$, $EC_{i,3} = FE_i + SE_{i,3} + PE_i$ and $GE_i = \frac{AE_i - PE_i}{AE_i}$, we can finally get $GE_i = \frac{(EC_{i,2} - EC_{i,3}) - SP(ET_{i,2} - ET_{i,3})}{(EC_{i,2} - EC_{i,1}) - SP(ET_{i,2} - ET_{i,1})}$. Then we can compute $\overline{GE}$ with all GE_i.

To prepare for this experiment, we implement a test APP using JNI technique to invoke random file reading and writing without caching data into various buffer mechanism. We also turn off irrelevant APPs and sensors and then run our test APP while measuring energy consumption.

Then we use Monsoon power monitor to observe SP, $EC_{i,j}$, $ET_{i,j}$ and compute $\overline{GE}$ based on our comparison model. We measure and compute $\overline{GE}$ with five repeated experiments and found that $\overline{GE}$ is roughly between 50% and 70%. Therefore, we can conclude that *Populus* saves 50%-70% less energy than dm-crypt.

6. Related Work

Popular and secure disk encryption software includes dm-crypt (for Linux and Android), BitLocker (for Windows), FileVault (for Mac OS X) and TrueCrypt (for Windows and Linux)[22]. They conduct encryption/decryption with tweakable scheme[12] and SPN-based block ciphers[13]. However, we found that tweakable scheme and SPN essentially lead to the energy overhead in disk encryption software and explained it in Section 2. As an attempt to improve efficiency, Crowley and Paul proposed Mercy, a lightweight disk encryption software[10]. Unfortunately, Fluhrer proved that Mercy is insecure in cryptography[11].

7. Conclusion

In this paper, we develop a kernel-level disk encryption software *Populus* to reduce the high energy consumption of disk encryption, which is critical for mobile devices. We observe that at most 98% of *Populus's* encryption/dycryption computation is input-free, which can be accomplished in advance during initialization, so *Populus* is energy-efficient for processing real-time encryption/decryption requests. We conduct cryptanalysis on *Populus* and find it is computationally secure when facing state-of-the-art cryptanalysis techniques. We also conduct energy consumption experiments and our experimental results show that *Populus* consumes 50%-70% less energy in comparison with dm-crypt.

References

1. G. Gualdi, A. Prati, R. Cucchiara, Video streaming for mobile video surveillance, Multimedia, IEEE Transactions on 10 (6) (2008) 1142–1154.
2. M. La Polla, F. Martinelli, D. Sgandurra, A survey on security for mobile devices, Communications Surveys & Tutorials, IEEE 15 (1) (2013) 446–471.
3. Wiki, Disk encryption theory, `https://en.wikipedia.org/wiki/Disk_encryption_theory`, [Online; accessed 2-January-2016] (2016).
4. V. Svajcer, sophos mobile security threat report (2014).
5. A. M. Zambrano, I. Perez, C. Palau, M. Esteve, Distributed sensor system for earthquake early warning based on the massive use of low cost accelerometers, Latin America Transactions, IEEE (Revista IEEE America Latina) 13 (1) (2015) 291–298.
6. J. Li, A. Badam, R. Chandra, S. Swanson, B. L. Worthington, Q. Zhang, On the energy overhead of mobile storage systems., in: FAST, 2014, pp. 105–118.
7. C. Xiao, W. Wang, N. Yang, L. Wang, A video sensing oriented speed adjustable fast multimedia encryption scheme and embedded system, in: Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE, IEEE, 2014, pp. 234–238.
8. A. Carroll, G. Heiser, An analysis of power consumption in a smartphone., in: USENIX annual technical conference, Vol. 14, Boston, MA, 2010.
9. F. Xia, C.-H. Hsu, X. Liu, H. Liu, F. Ding, W. Zhang, The power of smartphones, Multimedia Systems 21 (1) (2015) 87–101.
10. P. Crowley, Mercy: A fast large block cipher for disk sector encryption, in: Fast Software Encryption, Springer, 2001, pp. 49–63.
11. S. R. Fluhrer, Cryptanalysis of the mercy block cipher, in: Fast Software Encryption, Springer, 2002, pp. 28–36.
12. P. R. Shai Halevi, A tweakable enciphering mode, Springer Berlin Heidelberg (2003) 482–499.
13. C. E. Shannon, Communication theory of secrecy systems*, Bell system technical journal 28 (4) (1949) 656–715.
14. P. Rogaway, Nonce-based symmetric encryption, in: Fast Software Encryption, Springer, 2004, pp. 348–358.
15. D. J. Bernstein, The salsa20 family of stream ciphers, in: New stream cipher designs, Springer, 2008, pp. 84–97.
16. NIST, selects winner of secure hash algorithm (sha-3) competition, `http://www.nist.gov/itl/csd/sha-100212.cfm`, [Online; accessed 11-April-2015] (2012).
17. B. Acharya, S. K. Patra, G. Panda, Involutory, permuted and reiterative key matrix generation methods for hill cipher system, matrix 2 (2009) 1.
18. J. Overbey, W. Traves, J. Wojdylo, On the keyspace of the hill cipher, Cryptologia 29 (1) (2005) 59–72.
19. M. Eisenberg, Hill ciphers and modular linear algebra, November 3th.
20. Y. Hu, J. Lui, W. Hu, X. Ma, J. Li, X. Liang, Taming energy cost of disk encryption software on data-intensive mobile devices, arXiv preprint arXiv:1604.07543.
21. Monsoon power monitor, `http://www.msoon.com/LabEquipment/PowerMonitor/`.
22. Wiki, Disk encryption software, `https://en.wikipedia.org/wiki/Disk_encryption_software`, [Online; accessed 17-February-2016] (2015).