Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Taming energy cost of disk encryption software on data-intensive mobile devices

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

[a] *The Chinese University of Hong Kong, Hong Kong, China*
[b] *MOE KLINNS Lab, Xi'an Jiaotong University, Xi'an, Shaanxi, China*
[c] *Palo Alto Networks Inc., USA*

ARTICLE INFO

Article history:
Received 31 October 2016
Received in revised form 26 August 2017
Accepted 7 September 2017
Available online xxxx

Keywords:
Privacy protection
Disk encryption
Energy-efficient computing

ABSTRACT

Disk encryption software is frequently used to secure confidential data on mobile devices. However, it is notoriously challenging for disk encryption software to ensure its security in cryptography without involving significant energy overhead. To address the challenge, we design a both cryptographically secure and energy-efficient disk encryption software for mobile devices, Populus. On the one hand, Populus uses modular linear algebra and one-time pad technique to encrypt/decrypt sensitive data on mobile devices, thus ensuring its security in cryptograph. To illustrate, we prove Populus's semantic security. On the other hand, Populus is based on client–server pattern. Its client side works on the kernel layer of mobile devices powered by batteries, while its server side works on the application layer of computing devices powered by fixed electric power source. The server side helps the client side do the computation tasks unrelated to plaintext/ciphertext in the encryption/decryption process, therefore, the energy cost on mobile devices significantly declines. To demonstrate, we conduct energy consumption experiments on Populus and dm-crypt, a famous disk encryption software for Android and Linux mobile devices. The experimental results show that Populus consumes 50%–70% less energy than dm-crypt.

1. Introduction

In recent years, mobile devices (e.g., smartphones, smartwatches and mobile video surveillance devices) have become an integral part in our daily life. Meanwhile, mobile devices are usually facing profound security challenges [1], 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.

However, for data-intensive applications such as mobile video surveillance [5] and seismic monitor [6], the whole energy consumption of mobile devices highly rises after applying state-of-the-art 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 mobile devices when turning on their disk encryption software [7]. Worse, mobile devices are usually battery-powered in order to improve portability. For example, a mobile video surveillance device is equipped on a multi-rotor unmanned aerial vehicle, and battery becomes its sole power supply [5]. Due to mobile devices' limited battery capacity, state-of-the-art disk encryption software may strongly affect their normal usage.

After our careful analysis, we find that the *cipher* (i.e., the encryption/decryption module) used in disk encryption software is the main source of energy overhead. In particular, the cipher of disk encryption software contains many CPU and RAM operations, and disk encryption software has to conduct those operations many times for data-intensive applications. As a result, disk encryption software has to conduct a large amount of CPU and RAM operations, which causes significant energy overhead. To illustrate, consider mobile video surveillance devices which need to real-timely record and encrypt a large amount of video data [5]. According to our experiments, nearly 1/3 of the energy consumption comes from popular disk encryption software in mobile video surveillance.

In fact, the energy consumed by CPU and RAM operations tends to become more prominent than other sources such as

* Correspondence to: Room 111, Ho Sin Hang Engineering Building, The Chinese University of Hong Kong, Hong Kong, China.
E-mail addresses: huyang0905@126.com (Y. Hu), cslui@cse.cuhk.edu.hk (J.C.S. Lui), whu@paloaltonetworks.com (W. Hu), xma.cs@xjtu.edu.cn (X. Ma), jfli@sei.xjtu.edu.cn (J. Li), qingyuanxingsi@163.com (X. Liang).

https://doi.org/10.1016/j.future.2017.09.025
0167-739X/© 2017 Elsevier B.V. All rights reserved.

screen and network communication, especially when disk encryption software handles data-intensive tasks. About *six years ago*, about 45%–76% of daily energy consumption came from screen and GSM [8]. However, the distribution of energy consumption has been changed dramatically in recent years due to software/hardware optimization. A recent study [9] shows that for typical usage of mobile devices, only about 28% of energy consumption comes from screen and GSM, while 35% of energy consumption comes from CPU and RAM, and CPU and RAM become two largest energy consumption sources in mobile devices. Furthermore, both [8] and [9] measure energy consumption without enabling disk encryption software. Considering that existing disk encryption software owns many CPU and RAM operations, we believe that the percentage of the energy consumption on CPU and RAM may be much higher than 35% if mobile devices enable disk encryption software when handling data-intensive tasks. Li's experimental results [7] exactly verified it. Hence, to build a both energy-efficient and secure mobile system, reducing the energy consumption on disk encryption software is a rational starting point.

To reduce the 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 state-of-the-art *cryptographically secure* disk encryption software, the disk encryption software used in Linux and Android, also known as *dm-crypt*, 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.

In this paper, we design a both cryptographically secure and energy-efficient disk encryption software for mobile devices, *Populus*. *Populus* is built upon client–server pattern. Its client side is deployed on battery-powered mobile devices, and its server side is deployed on computing devices powered by fixed electric power source. Before encrypting/decrypting sensitive data, the client side requests the server side to do the *input-free* computation tasks in the encryption/decryption tasks. Here, 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. After accomplishing these tasks in the computing environment with enough power supply, the server side sends the computation results back to the client side. Finally, the client side accomplishes the residual computation in the encryption/decryption tasks.

In addition, to avoid energy overhead caused by frequent communication between the client side and the server side, the client side can reduce the request frequency by asking the server side to do the input-free computation tasks in the predictable future encryption/decryption tasks. For example, if a mobile device is going to encrypt x-GB data in the future y days, the client side can ask the server side to do input-free computation in the x-GB data encryption task in one request. Therefore, the client side will not need to send more requests in the future y days.

With the design above, *Populus* helps to save much energy on mobile devices, when the encryption/decryption process of *Populus* owns much input-free computation. However, the ciphers used in existing disk encryption software are based on *substitution–permutation network* (SPN), which only has a little input-free computation. For example, we find that the input-free computation of AES-CBC cipher accounts for at most 1% of all its computation.

To improve the proportion of the input-free computation without sacrificing *Populus's* security in cryptography, *Populus* uses modular linear algebra and one-time pad technique to encrypt/decrypt sensitive data on mobile devices. The server side generates *pseudo random numbers* (*PRNs*) and global matrices in an input-free manner and send their ciphertext to the client side. Then the client side decrypts them and uses them to construct *temporary* matrices and then uses carefully designed matrix multiplication to encrypt/decrypt disk sectors with those temporary matrices. In this way, *Populus* can save much energy on mobile devices without destroying its security in cryptography, because the encryption method with temporary matrices effectively defends against chosen-plaintext attack and almost 98% of its computation is input-free. In addition, *Populus* only needs to use 6.1%–7.8% extra storage space to store global keys, PRNs and etc.

We conduct security analysis on *Populus*. In particular, we prove that *Populus* is semantically secure against state-of-the-art chosen-plaintext attack methods. We also conduct a series of energy consumption experiments on both *Populus* and dm-crypt. Our experimental results indicate that *Populus* can save 50%–70% more energy than dm-crypt. The contributions of this work are as follows.

- 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 a both cryptographically secure and energy-efficient disk encryption software *Populus*.

The remainder of the paper is organized as follows. Section 2 introduces the threat model in disk encryption theory. In Section 3, we introduce the details on input-free computation. Section 4 presents our system *Populus* in detail. Then we show our cryptanalysis in Section 5 and present the experimental results of Populus's energy consumption in Section 6. Section 7 summarizes related work. Concluding remarks then follow.

2. Threat model

We assume that the disk encryption software manages disk sectors for secure storage. The user requests the disk encryption software to encrypt certain plaintext with a secret key. Then the disk encryption software allocates some disk sectors to the user and stores the ciphertext on these disk sectors. The attacker aims to obtain the plaintext with the following abilities. First, the attacker knows the encryption/decryption algorithm that the disk encryption software uses. Second, the attacker can read the data stored in any disk sectors that the disk encryption software manages. Third, the attacker can write arbitrary data to any disk sectors that the disk encryption system manages. Fourth, the attacker can request the disk encryption software to encrypt arbitrary data, and the disk encryption software allocates some unused disk sectors to the attacker in order to store the corresponding ciphertext. Fifth, the attacker can request the disk encryption software to decrypt the data in the disk sectors allocated to the attacker.

3. Input-free computation

In this section, we first explain why state-of-the-art disk encryption software is lack of input-free computation. Then we show how *Populus* improves the proportion of input-free computation.

3.1. Low input-free computation

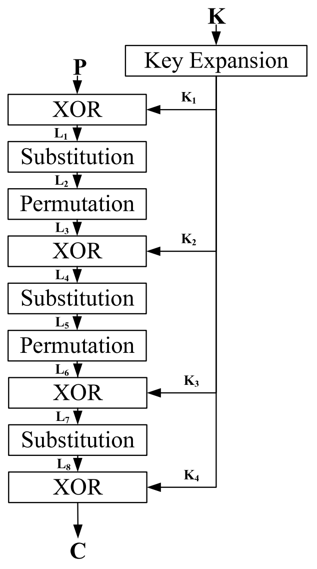
To explain the causes of low input-free computation, we should first understand the design considerations of disk encryption software. According to the threat model introduced in Section 2, we can infer that different disk sector should be encrypted with different key. Otherwise, the attacker can solve the plaintext by reading the ciphertext of the privacy from the disk sectors allocated to the user, writing the ciphertext to the disk sectors allocated to the attackers and requesting the disk encryption software to decrypt the ciphertext in the disk sectors allocated to the attacker. Hence, most existing disk encryption software uses the key provided by the user and disk sector ID to produce the keys of all disk sectors, which is known as *tweakable scheme* [12].

However, tweakable scheme is not absolutely secure in cryptography. In detail, since tweakable scheme allows the attacker to get multiple (plaintext, ciphertext) pairs in the same disk sector, the attacker can conduct *chosen-plaintext attack* (*CPA*) to solve the privacy the user stores. One of the most effective methods to defend against CPA is to apply *substitution–permutation network* (*SPN*) [13] to tweakable-based block cipher in disk encryption software. SPN separates one encryption/decryption process into several rounds, and each round uses substitution boxes, permutation boxes and round key to diffuse and confuse plaintext. SPN achieves Shannon's confusion/diffusion properties and can highly reduce the probability that CPA succeeds in a reasonable computational complexity [13].

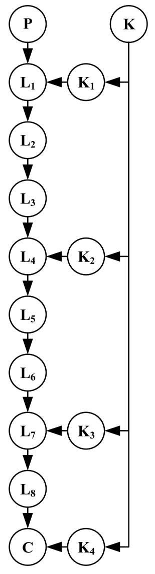
Despite the merits of SPN, we find that nearly all SPN-based block ciphers only have a little input-free computation because their core components (i.e., *substitution box* and *permutation box*) directly or indirectly rely on inputs. Fig. 1(a) depicts an example of SPN-based cipher, where P denotes the plaintext, K is the key of a disk sector, $K_1, \dots, K_4$ are four round keys produced by key expansion, C is the ciphertext and $L_1, \dots, L_8$ are the middle results when the SPN processes P. To illuminate why this cipher lacks input-free computation, we describe its data dependency in Fig. 1(b). Each node denotes one of $P, C, K, K_1, \dots, K_4, L_1, \dots, L_8$. We draw an arrow from a node A to another node B only when the computation of B must be conducted after the computation of A. For example, computing L_1 must happen before computing L_2, because L_1 and L_2 are separately the input and the output of the same substitution box. As P is the input, any computation which directly or indirectly relies on P does not belong to input-free computation. Therefore, the computation of $K_1, \dots, K_4$ is input-free while the computation of $L_1, \dots, L_8, C$ is not input-free. Since the computation of $K_1, \dots, K_4$ is far less than that of $L_1, \dots, L_8, C$, preprocessing the input-free computation of the cipher has little effect on saving energy.

3.2. How to improve input-free computation

To improve input-free computation, 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 the key provided by the user and *i*; (b) use the temporary key and a light-weight block cipher to accomplish *i*th encryption. Our design has three advantages. First, it makes attackers hard to get multiple (plaintext, ciphertext) pairs sharing same key so as to mitigate the threat from CPA. Second, nearly all procedures in (a) are input-free. Third, SPN becomes unnecessary in (b), which gives us more freedom to design a light-weight block cipher with much input-free computation. As a trade-off, our scheme needs extra storage space for the results of 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 4, we complement details regarding the design and implementation of *Populus*.



(a) An example of SPN-based cipher.



(b) Data dependencies in the SPN-based cipher.

Fig. 1. An illustration that SPN-based ciphers are lack of input-free computation.

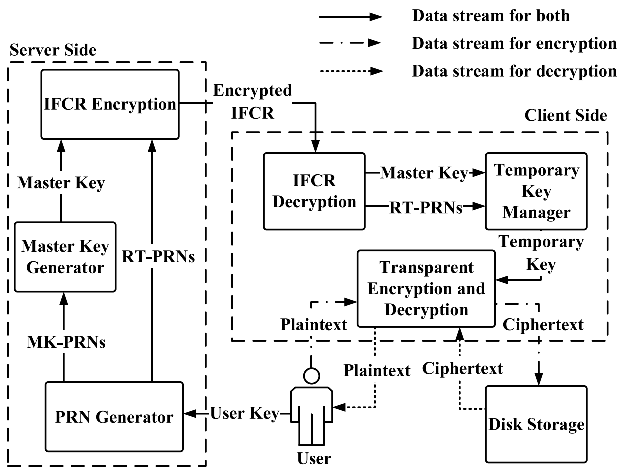


Fig. 2. Overview of Populus.

4. Populus: An energy-saving disk encryption software system

The overview of *Populus* is shown in Fig. 2. *Populus* consists of two parts: *server side* and *client side*. The server side accomplishes all input-free computation and sends its result to the client side, 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. 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.

4.1. Server side

The server side 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.

4.1.1. 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, and 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*.

4.1.2. Master key generator

Populus generates master key using MK-PRNs. We define that a square matrix A is *modular invertible* when there exists a matrix B such that $AB \bmod 2^{64} = I$, 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. For simplicity, the modular inverse of A is denoted by A^{-1} in this paper. We define the master key as $U = (U^{(1)}, \dots, U^{(125)})$, where each $U^{(i)} = \begin{pmatrix} u^{(i)}_{1,1} & u^{(i)}_{1,2} \\ u^{(i)}_{2,1} & u^{(i)}_{2,2} \end{pmatrix}$, $1 \le i \le 125$, 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 $U^{(1)}, \dots, U^{(125)}$ 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. Using the method proposed by [18] , we can compute that the number of all possible U is about 3.38×10^{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.

4.1.3. IFCR encryption and decryption

In our threat model, attackers can physically access disk. To avoid attackers reading the plaintext of IFCR (i.e., RT-PRNs and master key) from disk, *Populus* encrypts them and then sends them to the client side. During real-time encryption/decryption, *Populus* decrypts the master key and RT-PRNs from disk. Later, we will introduce more details in Section 4.3.

4.2. Client side

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

4.2.1. Transparent encryption and decryption

Our transparent encryption and decryption are based on matrix multiplication in *modular linear algebra*. In cryptography, matrix multiplication has achieved Shannon's diffusion [13], and it dissipates statistical structure of the plaintext into long-range statistics of the ciphertext to thwart cryptanalysis based on statistical analysis [19]. However, matrix multiplication is usually computationally intensive. For example, a general matrix multiplication between a 64×64 matrix and a 64×1 matrix requires at least $64 \times 64 + 64 \times 63 + 128 = 8256$ 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 $i \in \{1, \dots, 125\}$, I_i is the i-dimensional identity matrix and $M^{(i)}$ is a 2×2 modular invertible matrix. Then *Populus* computes $H^{(125)} \dots H^{(1)}P$ as encryption or $(H^{(1)})^{-1} \dots (H^{(125)})^{-1}C$ 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. Given that $H^{(i)}$ is a sparse matrix, 125 64-dimensional matrix multiplications can be simplified to 125 2-dimensional matrix multiplications. The simplified encryption and decryption only consist of $125 \times (2 \times 2 + 2 \times 1) + 128 = 868$ operations. Now, we describe our transparent encryption and decryption in more detail. Fig. 3 presents its full view. Let $P = (P^{(1)}, \dots, P^{(64)})^T$, $C = (C^{(1)}, \dots, C^{(64)})^T$ and $M = (M^{(1)}, \dots, M^{(125)})$ denote its plaintext, ciphertext and temporary key respectively, where $P^{(i)}$ is the ith number in the plaintext, $C^{(i)}$ is the ith number in the ciphertext and $M^{(i)} = \begin{pmatrix} m^{(i)}_{1,1} & m^{(i)}_{1,2} \\ m^{(i)}_{2,1} & m^{(i)}_{2,2} \end{pmatrix}$ is the ith 2×2 matrix in M. For simplicity, we use the notation $[m]_n$ to denote the function $m \bmod n$, i.e., $[m]_n = m \bmod n$. The encryption function $E(P, M)$ works as follows. We first set $\beta^{(1,j)} = P^{(j)}$ and then iteratively compute $\beta^{(i+1,j)}$, $1 \le i \le 125$, $1 \le j \le 64$, as

$$\beta^{(i+1,j)} = \begin{cases} [\beta^{(i,j)} m^{(i)}_{1,1} + \beta^{(i,j+1)} m^{(i)}_{1,2}]_{2^{64}}, & j = i,\ 126 - i \\ [\beta^{(i,j-1)} m^{(i)}_{2,1} + \beta^{(i,j)} m^{(i)}_{2,2}]_{2^{64}}, & j = i+1,\ 127 - i \\ \beta_{i,j}, & \text{otherwise.} \end{cases}$$

Finally, we set $E(P, M) = (\beta^{(126,1)}, \dots, \beta^{(126,64)})^T$.

The decryption $D(C, M)$ function works as follows. Let $(M^{(i)})^{-1} = \begin{pmatrix} l^{(i)}_{1,1} & l^{(i)}_{1,2} \\ l^{(i)}_{2,1} & l^{(i)}_{2,2} \end{pmatrix}$ and $k = 126 - i$. We set $\gamma^{(1,j)} = C^{(j)}$ and then iteratively compute $\gamma^{(i+1,j)}$, $1 \le i \le 125$, $1 \le j \le 64$, as

$$\gamma^{(i+1,j)} = \begin{cases} [\gamma^{(i,j)} l^{(k)}_{1,1} + \gamma^{(i,j+1)} l^{(k)}_{1,2}]_{2^{64}}, & j = i,\ 126 - i \\ [\gamma^{(i,j-1)} l^{(k)}_{2,1} + \gamma^{(i,j)} l^{(k)}_{2,2}]_{2^{64}}, & j = i+1,\ 127 - i \\ \gamma^{(i,j)}, & \text{otherwise.} \end{cases}$$

Finally, we set $D(C, M) = (\gamma^{(126,1)}, \dots, \gamma^{(126,64)})^T$.

4.2.2. Temporary key manager

To avoid chosen-plaintext attack, M cannot be reused many times. In particular, if an attacker can get 64 (plaintext, ciphertext) pairs corresponding to the same temporary key M, the attacker can directly solve M with linear transformation (see Section 5 for more detail). To solve this problem, we design a temporary key manager with a low reusing rate. For each time of the disk sector encryption, *Populus* randomly selects one temporary key from a large key space, hence lowering the probability of reusing the same temporary key and making attackers hard to accomplish chosen-plaintext attack. In addition, the storage cost of temporary keys can be controlled in a low level.

Next, we give a formal description on the temporary key manager. *Populus* only stores U and RT-PRNs $R = (R_1, \dots, R_{2d})$ instead of d temporary keys. When conducting jth encryption, *Populus* computes the temporary key based on U, R_{2j-1} and R_{2j}. This way, *Populus* only costs about 16 bytes for each temporary key when d is large. Then we introduce how to produce the temporary key M with U, R_{2j-1}, R_{2j} for jth encryption.

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

This method makes sure that $M^{(i)}$ is modular invertible, therefore, $M^{(i)}$ can be used in our transparent encryption/decryption.

Theorem 1. *$M^{(i)}$ is modular invertible.*

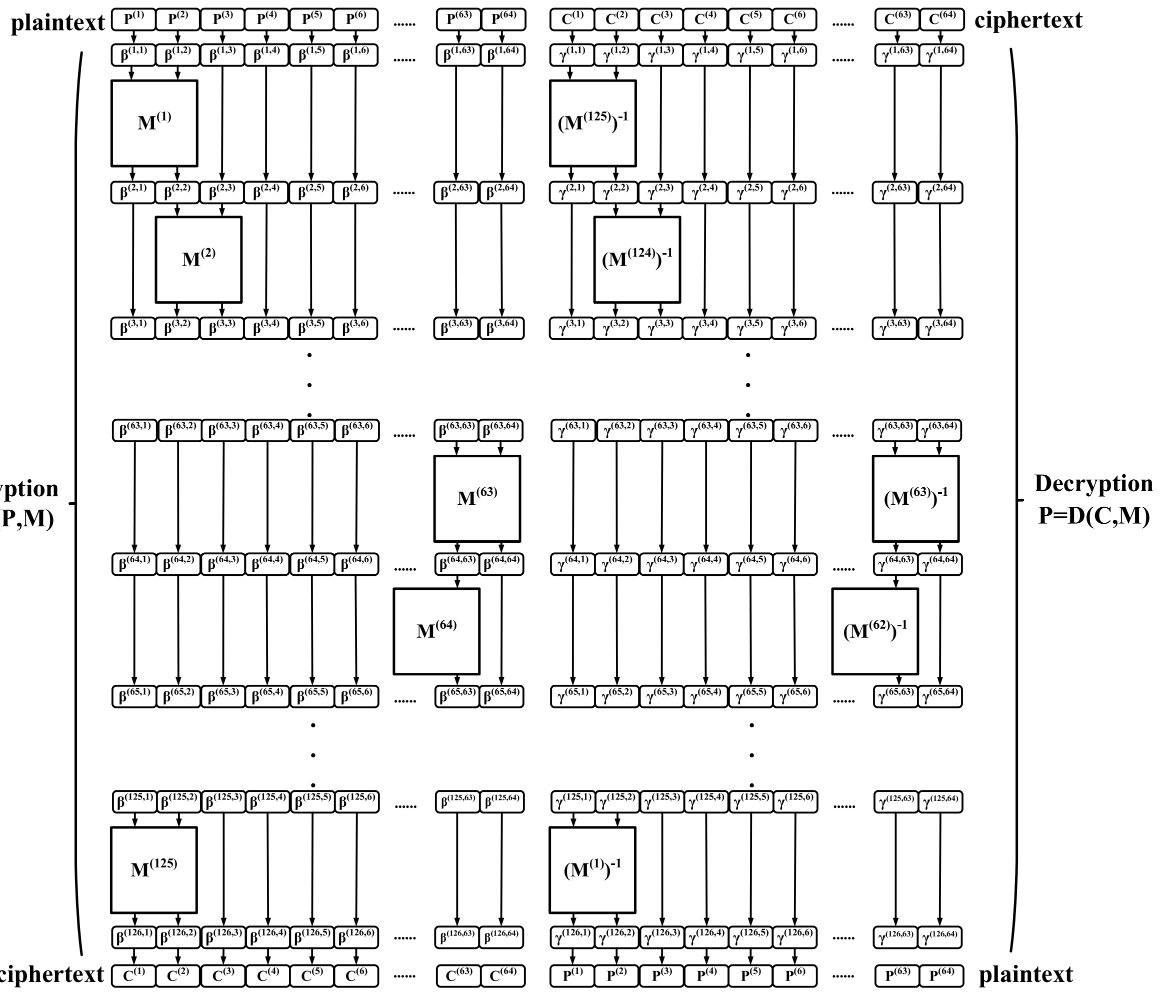


Fig. 3. An illustration of transparent encryption and decryption.

Proof. From [20], one can claim that $M^{(i)}$ is modular invertible if and only if $|M^{(i)}|$ and 2^{64} are co-prime, where $|M^{(i)}|$ denotes the determinant of matrix $M^{(i)}$. Therefore, $M^{(i)}$ is modular invertible when $|M^{(i)}|$ is an odd number. Next, we prove $|M^{(i)}|$ is an odd number. From the definition of $m_{p,q}^{(i)}$, we can easily find that $m_{p,q}^{(i)}$ and $u_{p,q}^{(i)}$ have the same parity for any $p, q \in \{1, 2\}$. Thus, $|M^{(i)}| = m_{1,1}^{(i)}m_{2,2}^{(i)} - m_{1,2}^{(i)}m_{2,1}^{(i)}$ and $|U^{(i)}| = u_{1,1}^{(i)}u_{2,2}^{(i)} - u_{1,2}^{(i)}u_{2,1}^{(i)}$ have the same parity. $U^{(i)}$ is modular invertible, so we know $|U^{(i)}|$ and 2^{64} are co-prime. Thus, $|U^{(i)}|$ and $|M^{(i)}|$ are both odd numbers.

After applying our method, only small storage space of RT-PRNs is able to satisfy most 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 $lt/256$ pseudo random numbers in RT-PRNs. For example, as for a smartphone, we let $t = 4$ and $l = 2^{31}$ so that only 256 MB are required to store 2^{25} pseudo random numbers.

4.3. Iterative encryption and decryption on IFCR

In Section 4.1.3, we have briefly introduced the function of IFCR (i.e., the master key and RT-PRNs) 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, all encrypted IFCR should be decrypted for one transparent encryption in the worst case, which obviously costs lots of energy when the size of IFCR is large.

We propose the *iterative encryption/decryption* to reduce the energy cost on IFCR decryption. To protect users' sensitive data in some disk sectors, *Populus* produces IFCR Ω_1 and encrypt those disk sectors with Ω_1. Then, *Populus* produces another IFCR Ω_2 whose size is smaller than Ω_1, encrypt Ω_1 with Ω_2 through our transparent encryption method and store the ciphertext of Ω_1 to the disk. We continue this iteration n times when the size of Ω_n is small enough. Finally, we use a user key Ω_u to encrypt Ω_n with existing block ciphers such as AES-CBC and store the ciphertext of Ω_n to the disk. Note that the user key Ω_u is not stored in disk. The user needs to remember the user key and input it to our system during its initialization. This way, all content in the disk is encrypted, so attackers cannot directly read the plaintext of those keys from the disk.

Next, we describe iterative encryption/decryption in detail. Considering that *Populus* only needs one new master key (4000 bytes) and $2k$ new RT-PRNs ($16k$ bytes) to encrypt $512k$-byte data, *Populus* iteratively encrypts them as follows: (a) if $4000 + 16k \geq 512k$ ($k \leq 9$), *Populus* directly encrypts $512k$-byte data with a SPN-based cipher (e.g., AES-CBC); (b) if $4000 + 16k < 512k$ ($k > 9$), *Populus* produces one new master key and $\lceil (16k + 4000)/512 \rceil$ new RT-PRNs and use them to encrypt the $512k$-byte data with our proposed transparent encryption method. Then *Populus* encrypts the new master key and new RT-PRNs with step (a) and (b). As for *iterative decryption*, just reverse the whole process of iterative encryption. This way, iterative encryption/decryption can protect IFCR with less computation than existing block cipher, which plays a important role in saving energy on mobile devices.

To illustrate that the iterative encryption/decryption helps to save energy on mobile devices, we give a computational complexity analysis on iterative encryption/decryption and AES-CBC, a baseline block cipher. Let $f(n)$ denote the computational complexity of decrypting one of the n encrypted disk sectors for sensitive data when using iterative encryption/decryption to protect the IFRC for the n encrypted disk sectors. When $n \le 9, f(n) = O(1)$. When $n > 9$, we should first iteratively decrypt its corresponding IFCR which occupies $\lceil (16n + 4000)/512 \rceil$ disk sectors. The master key has 4000 bytes stored in eight disk sectors and two pseudo random numbers are stored in one disk sector. Therefore, we should iteratively decrypt nine of the $\lceil (16n + 4000)/512 \rceil$ disk sectors. So we can get:

$$f(n) = \begin{cases} 9f(\lceil (16n+4000)/512 \rceil) + O(1), & n > 9 \\ O(1), & n \le 9 \end{cases}$$

Using master theorem, we can obtain $f(n) = O(log(n))$. Let $g(n)$ denote the computational complexity of decrypting one of the n encrypted disk sectors for sensitive data when using AES-CBC to protect the IFRC for the n encrypted disk sectors. Since decrypting one disk sector needs its corresponding IFCR, we should use AES-CBC to decrypt its corresponding IFCR which occupies $\lceil (16n + 4000)/512 \rceil$ disk sectors. So we can get $g(n) = O(\lceil (16n + 4000)/512 \rceil) = O(n)$. Comparing $f(n)$ with $g(n)$, we can see iterative encryption/decryption has a smaller computation complexity, thus saving more energy under most data-intensive scenarios.

One limitation of our iterative encryption/decryption is that it has to cost extra storage due to the newly produced IFCRs. But, according to our analysis, its storage overhead is slight and tolerable in most data-intensive scenarios. To illustrate, let us compute and analyze the storage cost of IFCR. Let $h(n)$ denote the storage cost (i.e., number of disk sectors) of all IFCRs when using iterative encryption to protect the IFRC for the encrypted n-sector sensitive data. When $n \le 9$, $h(n) = 0$. When $n > 9$, we need to produce new IFCR which occupies $\lceil (16n+4000)/512 \rceil$ disk sectors and then use iterative encryption to protect the newly produced IFCR. Therefore, we can get:

$$h(n) = \begin{cases} h(\lceil (16n+4000)/512 \rceil) + \lceil (16n+4000)/512 \rceil, & n > 9 \\ 0, & n \le 9 \end{cases}$$

When $n = 32^k b + 9$, $k \ge 0$, $b \in \{1, \ldots, 31\}$,

$$\begin{aligned} h(32^k b + 9) &\le h(32^{k-1} b + 9) + 32^{k-1} b + 9 \\ &\le (32^k - 1)b/(32 - 1) + 9k + h(9 + b) \\ &\le (32^k - 1)b/(32 - 1) + 9k + h(40) \\ &\le (n - 9 - b)/31 + 9log_{32}(n - 9) - 9log_{32}(b) + 19 \\ &\le (n - 10)/31 + 9log_{32}(n - 9) - 9log_{32}(1) + 19 \\ &< n/31 + 9log_{32}(n) + 19 \end{aligned}$$

When $32^k b + 9 < n < 32^k (b + 1) + 9$,

$$\begin{aligned} h(n) &\le h(32^k (b + 1) + 9) \\ &\le (32^k - 1)(b + 1)/(32 - 1) + 9k + h(9 + b) \\ &\le (32^k - 1)(b + 1)/(32 - 1) + 9k + h(40) \\ &\le ((n - 9)(b + 1)/b - b - 1)/31 + 9log_{32}(n - 9) \\ &\quad - 9log_{32}(b) + 19 \\ &\le (2(n - 9) - 2)/31 + 9log_{32}(n - 9) - 9log_{32}(1) + 19 \\ &< 2n/31 + 9log_{32}(n) + 19 \end{aligned}$$

Since $0 < n/31 + 9log_{32}(n) + 19 < 2n/31 + 9log_{32}(n) + 19$, we can finally get $h(n) < 2n/31 + 9log_{32}(n) + 19$.

We use a widely-used storage cost metric $h(n)/(h(n) + n)$, the percentage of IFCR's storage cost among total storage cost. We compute $h(n)/(h(n) + n)$ when $n \in \{2^{25}, 2^{26}, \ldots, 2^{45}\}$, which restrains the storage space of encrypted sensitive data between 1MB to 1PB. As a result, only no more than 7.8% of the total storage cost comes from IFCR, when $n \in \{2^{15}, 2^{26}, \ldots, 2^{45}\}$. Moreover, when n grows larger, $h(n)/(h(n) + n)$ will become closer to

$$\lim_{n \to +\infty} h(n)/(h(n) + n) = 2/33 \approx 6.1\%$$

Sacrificing 6.1%–7.8% extra storage cost for saving 50%–70% energy is usually cost-efficient for mobile devices in data-intensive scenarios.

5. Security analysis of populus

In this section, we analyze *Populus's* security in cryptography based on semantic security model in cryptography. First, we introduce the security model and notations. In particular, we define a encryption scheme and a *IND-CPA game*. Then we prove the attacker cannot win the game with the best effective method known today, indicating that *Populus* is a semantically secure cryptosystem in practice.

5.1. Security model and notations

Semantic security model has been widely used in the security proofs of cryptosystems such as [21,22]. This model consists of the encryption scheme and IND-CPA game. The encryption scheme is a tuple of key space, one encryption method and one decryption method. One should choose a key randomly from the key space and then use the encryption/decryption method and the key to encrypt/decrypt data. In IND-CPA game, the attacker has two groups of plaintexts. The attacker sends these two groups of plaintexts to the challenger. Then the challenger chooses one of them randomly, uses the encryption scheme to encrypt the plaintexts in the group and sends its ciphertext back to the attacker. Finally, the attacker determines which group the challenger chooses according to the two groups of plaintexts and received the ciphertext. If the attacker cannot correctly determine this with high success probability and a reasonable computational complexity, then we say the encryption scheme is *semantically secure against CPA* or *IND-CPA* secure.

Now we define the encryption scheme for the security proof of *Populus*.

Definition 1 (*Encryption Scheme of Populus*)**.** Let $\Pi = (T, E, D)$ denote the encryption scheme of Populus, where T denotes the space of temporary keys, $E : \{0, 1\}^{4096} \times T \to \{0, 1\}^{4096}$ denotes the encryption method and $D : \{0, 1\}^{4096} \times T \to \{0, 1\}^{4096}$ denotes the decryption method.

Now we define a IND-CPA game for the security proof of *Populus*. In fact, there are two widely accepted definitions for IND-CPA game: the concrete definition and the asymptotic definition. The concrete definition assumes that the key length is fixed, while the asymptotic definition allows varying key length. Since the key length of *Populus* is fixed, we define our IND-CPA game in a concrete manner. For simplicity, we use $A \overset{r}{\leftarrow} B$ to denote that the atom A is randomly chosen from the set B.

Definition 2 (*IND-CPA Game*)**.** Let $\Gamma^{\Pi}_{Adv} : \{0, 1\} \to \{0, 1\}$ denote the IND-CPA game. In Γ^{Π}_{Adv}, an attacker and a challenger interact with each other with the following steps:

Step 1. The attacker chooses two groups of plaintexts denoted by $P_{1:\theta} = (P_1, \ldots, P_\theta)$ and $P'_{1:\theta} = (P'_1, \ldots, P'_\theta)$, where $P_1 \ne \ldots \ne P_\theta$, $P'_1 \ne \ldots \ne P'_\theta$ and $P_i, P'_i \in \{0, 1\}^{4096}$. Then the attacker sends $P_{1:\theta}$ and $P'_{1:\theta}$ to the challenger.

Step 2. The challenger sends $Y_{1:\theta} = (E(X_1, M_1), \ldots, E(X_\theta, M_\theta))$ to the attacker, where $X_{1:\theta} = P_{1:\theta}$ when the game input is 0, $X_{1:\theta} = P'_{1:\theta}$ when the game input is 1 and $M_{1:\theta} \overset{r}{\leftarrow} T^\theta$.

Step 3. The attacker uses a distinguisher method $Adv : \{0,1\}^{4096} \rightarrow \{0,1\}$ to determine which group the challenger chooses (i.e., 0 for $P_{1:\theta}$ and 1 for $P'_{1:\theta}$) and return $Adv(Y_{1:\theta})$.

We say Π is (t, ϵ, θ) semantically secure against Adv if and only if the computational complexity of Adv is not more than t and

$$|\mathcal{P}(\Gamma^{\Pi}_{Adv}(0) = 1) - \mathcal{P}(\Gamma^{\Pi}_{Adv}(1) = 1)| \le \epsilon.$$

If $t = 2^{80}$, $\epsilon = 2^{-60}$ and $\theta \le t$ for Adv, we say Π is semantically secure or IND-CPA secure against Adv in practice.

In addition, when discussing *Populus's* security, we assume that *Populus's* pseudo random number generator is secure in cryptography. We do not discuss attack techniques beyond cryptography such as DMA-based attack, cold boot attack and evil maid attack.

5.2. Security proof of populus

Now we prove the attacker cannot win the IND-CPA game with the most efficient Adv methods known today. Since *Populus* is based on matrix multiplication, one of the most efficient Adv to the best of our knowledge is to exploit the algebraic properties in linear system. For example, $C_1 = LP_1, \dots, C_n = LP_n$ where L is $n \times n$ matrix, P_i, C_i are $n \times 1$ matrices or vectors and $P_1 \neq \dots \neq P_n$. Given $[C_1, \dots, C_n] = L[P_1, \dots, P_n]$, the attacker can solve L by computing

$$L = [C_1, \dots, C_n][P_1, \dots, P_n]^{-1}.$$

Similarly, the attacker can solve a temporary key of *Populus* if the attacker obtains 64 (plaintext, ciphertext) pairs sharing the same temporary key. With a temporary key, the attacker can decrypt its corresponding ciphertext received from the challenger and can use the decryption result to determine which group the challenger chooses.

Now, we formally describe the attack method. For simplicity, we reuse the notations in IND-CPA game. The attack method consists of two steps. First, the attacker lets $P_1 = P'_1, \dots, P_r = P'_r$ $(64 \le r < \theta)$ and then send $P_{1:\theta}, P'_{1:\theta}$ to *Populus*. Then *Populus* encrypts either $P_{1:\theta}$ or $P'_{1:\theta}$ with a set of temporary keys $M_{1:\theta}$ and sends the encryption results $Y_{1:\theta}$ back to the attacker. As $P_i = P'_i$ $(i \in \{1, \dots, r\})$, $Y_i = E(P_i, M_i)$ $(i \in \{1, \dots, r\})$, no matter which plaintext group (i.e., $P_{1:\theta}$ or $P'_{1:\theta}$) *Populus* chooses to encrypt. This way, the attacker can get r (plaintext, ciphertext) pairs $(P_1, C_1), \dots, (P_r, C_r)$, where $C_i = E(P_i, M_i)$ $(i \in \{1, \dots, r\})$. Second, the attacker constructs a distinguisher method Adv with $(P_1, C_1), \dots, (P_r, C_r)$ as follows. For every possible $Y_{1:\theta}$,

$$Adv(Y_{1:\theta}) = \begin{cases} 0, & event\ \mathcal{V}(Y_{1:\theta})\ happens, \\ 1, & otherwise. \end{cases}$$

Here, $\mathcal{V}(Y_{1:\theta})$ is the event that there exists 64 distinct temporary keys $M_{i_1}, \dots, M_{i_{64}}$ selected from $M_1, \dots, M_r$ and $\mu \in \{r+1, \dots, \theta\}$ satisfying

$$M_{i_1} = \dots = M_{i_{64}} = M_\mu,$$
$$[C_{i_1}, \dots, C_{i_{64}}][P_{i_1}, \dots, P_{i_{64}}]^{-1}P_\mu = Y_\mu.$$

Next, we demonstrate the key process of computing the upper bound of $|\mathcal{P}(\Gamma^{\Pi}_{Adv}(0) = 1) - \mathcal{P}(\Gamma^{\Pi}_{Adv}(1) = 1)|$ using following two lemmas and one theorem.

Lemma 1. $|\mathcal{P}(\Gamma^{\Pi}_{Adv}(0) = 1) - \mathcal{P}(\Gamma^{\Pi}_{Adv}(1) = 1)| \le \max \mathcal{P}(\mathcal{V}(C_{1:\theta}))$.

Proof. With *Bayes' theorem*, we can obtain the following equation.

$$\begin{aligned} & |\mathcal{P}(\Gamma^{\Pi}_{Adv}(0) = 1) - \mathcal{P}(\Gamma^{\Pi}_{Adv}(1) = 1)| \\ = & |\mathcal{P}(Adv(C_{1:\theta}) = 1) - \mathcal{P}(Adv(C'_{1:\theta}) = 1)| \\ = & |\mathcal{P}(Adv(C_{1:\theta}) = 0|\mathcal{V}(C_{1:\theta}))\mathcal{P}(\mathcal{V}(C_{1:\theta})) \\ & + \mathcal{P}(Adv(C_{1:\theta}) = 0|\neg\mathcal{V}(C_{1:\theta}))\mathcal{P}(\neg\mathcal{V}(C_{1:\theta})) \\ & - \mathcal{P}(Adv(C'_{1:\theta}) = 1|\mathcal{V}(C'_{1:\theta}))\mathcal{P}(\mathcal{V}(C'_{1:\theta})) \\ & - \mathcal{P}(Adv(C'_{1:\theta}) = 1|\neg\mathcal{V}(C'_{1:\theta}))\mathcal{P}(\neg\mathcal{V}(C'_{1:\theta}))|. \end{aligned}$$

Since $\mathcal{P}(Adv(C_{1:\theta}) = 0|\mathcal{V}(C_{1:\theta})) = \mathcal{P}(Adv(C'_{1:\theta}) = 1|\mathcal{V}(C'_{1:\theta})) = 1$ and $\mathcal{P}(Adv(C_{1:\theta}) = 0|\neg\mathcal{V}(C_{1:\theta})) = \mathcal{P}(Adv(C'_{1:\theta}) = 1|\neg\mathcal{V}(C'_{1:\theta})) = 0$, we can have

$$\begin{aligned} & |\mathcal{P}(Adv(C_{1:\theta}) = 1) - \mathcal{P}(Adv(C'_{1:\theta}) = 1)| \\ = & |\mathcal{P}(\mathcal{V}(C_{1:\theta})) - \mathcal{P}(\mathcal{V}(C'_{1:\theta}))| \\ \le & \max \mathcal{P}(\mathcal{V}(C_{1:\theta})). \end{aligned}$$

Lemma 1 implies that we only need to compute $\max \mathcal{P}(\mathcal{V}(C_{1:\theta}))$ instead of directly computing the upperbound of $|\mathcal{P}(Adv(C_{1:\theta}) = 1) - \mathcal{P}(Adv(C'_{1:\theta}) = 1)|$. The following lemma is on computing $\max \mathcal{P}(\mathcal{V}(C_{1:\theta}))$.

Lemma 2. *When* $r < 2^{122}$, *we have*

$$\mathcal{P}(\mathcal{V}(C_{1:\theta})) \le (\theta - r)e^{-r\mathcal{T}(\frac{64}{r}, \frac{1}{2^{128}})},$$

where $\mathcal{T}(x, y) = xlog(\frac{x}{y}) + (1 - x)log(\frac{1-x}{1-y})$.

Proof. Let $\alpha_{l,\mu}(l \in \{0, \dots, r\}, \mu \in \{r+1, \dots, \theta\})$ denote the event that there are exactly l matrices in $M_{1:r}$ which are equal to M_μ. With *Inclusion–Exclusion Principle*, we have

$$\mathcal{P}(\mathcal{V}(C_{1:\theta})) \le \sum_{\mu=r+1}^{\theta}(1 - \sum_{l=0}^{63}\tbinom{r}{l}\mathcal{P}(\alpha_{l,\mu})).$$

Next, we compute $\mathcal{P}(\alpha(j))$. Since *Populus* uses a master key and two RT-PRNs (128 bits in total) to produce a temporary key, we can get that $\mathcal{P}(M_i = M_\mu) = \frac{1}{2^{128}}$. So we can get $\alpha_{l,\mu} \sim B(r, \frac{1}{2^{128}})$ where B denotes the binomial distribution. Using the *Chernoff Bound*, we have

$$1 - \sum_{l=0}^{63}\tbinom{r}{l}\mathcal{P}(\alpha_{l,\mu}) \le e^{-r\mathcal{T}(\frac{64}{r}, \frac{1}{2^{128}})},$$

when $\frac{1}{2^{128}} < \frac{64}{r} < 1$. Finally, we have

$$\begin{aligned} \mathcal{P}(\mathcal{V}(C_{1:\theta})) & \le \sum_{\mu=r+1}^{\theta}(e^{-r\mathcal{T}(\frac{64}{r}, \frac{1}{2^{128}})}) \\ & = (\theta - r)e^{-r\mathcal{T}(\frac{64}{r}, \frac{1}{2^{128}})} \end{aligned}$$

when $r < 2^{122}$.

Theorem 2. *Populus is* $(t, (\theta - r)e^{-t\mathcal{T}(\frac{64}{t}, \frac{1}{2^{128}})}, \theta)$ *semantically secure against the Adv when* $r < 2^{122}$.

Proof. From Lemmas 1 and 2, we can get $|\mathcal{P}(\Gamma^{\Pi}_{Adv}(0) = 1) - \mathcal{P}(\Gamma^{\Pi}_{Adv}(1) = 1)| \le \max \mathcal{P}(\mathcal{V}(C_{1:\theta})) \le (\theta - r)e^{-r\mathcal{T}(\frac{64}{t}, \frac{1}{2^{128}})}$ when $r < 2^{122}$. Hence, *Populus* is $(t, (\theta - r)e^{-t\mathcal{T}(\frac{64}{t}, \frac{1}{2^{128}})}, \theta)$ semantically secure against the Adv when $r < 2^{122}$.

Using the scientific computational software *Wolfram Mathematica*, we get $(\theta - r)e^{-t\mathcal{T}(\frac{64}{t}, \frac{1}{2^{128}})} \ll 2^{-60}$ when $r < \theta \le t = 2^{80}$, indicating that Populus is semantically secure against the Adv in practice.

6. Energy consumption evaluation

In this section, we use *Monsoon power monitor* to measure energy consumption of the whole mobile device and estimate the energy cost of disk encryption software. We choose Google Nexus 4 smartphone with Android 5.0 OS as our test mobile device. We use dm-crypt as the baseline, since it is widely used in mobile devices.

6.1. Evaluation on the typical usage for mobile device

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

We 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 scenarios: video recording, video playing, data sending through WIFI and data receiving through WIFI. As for video playing and recording, the video format is mp4, and the video resolution is 480×270, and the choices of video length are 50 min, 100 min, 150 min and 200 min, and the video quality is of high definition. As for WIFI network, the choices of transferred data size are 256 MB, 512 MB, 768 MB, . . ., 2048 MB.

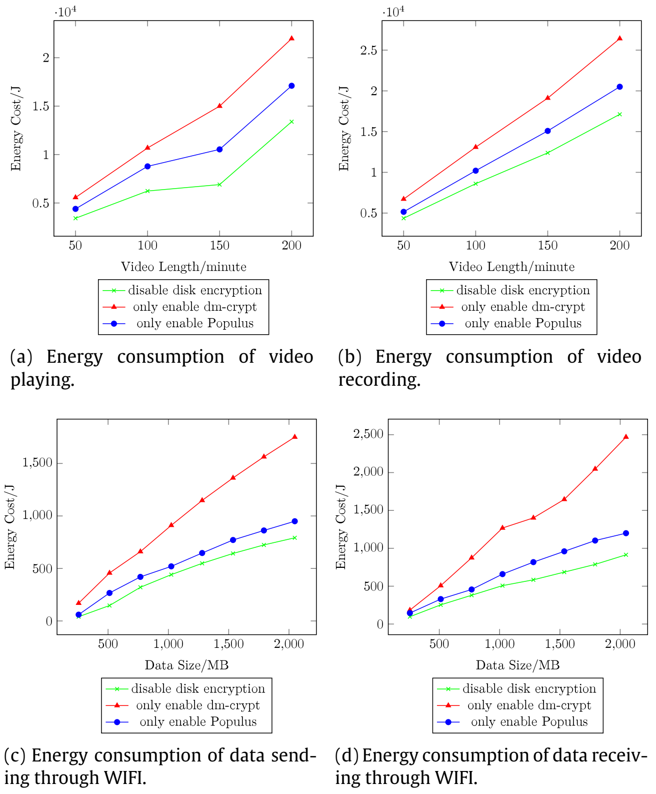
Then we introduce our experimental results separately. Video playing is a common function for handheld mobile devices such as smartphone, and its energy consumption is depicted in Fig. 4(a). Note that when playing an encrypted video, the disk encryption software should decrypt it in advance so that part of energy consumption comes from *Populus* or dm-crypt if they are enabled. As can be seen, nearly 1/2 of the whole energy consumption comes from dm-crypt, and *Populus* can reduce the proportion to nearly 1/4.

We also present experimental results of video recording in Fig. 4(b), as video recording on mobile devices is widely used in personal life, industry and military (e.g., mobile video surveillance). When recording a video, the disk encryption software encrypts the video data 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. 4(c) demonstrates mobile device's energy consumption when it sends data to another device through WIFI network. The data has been encrypted by disk encryption software in advance so that the energy cost by 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. 4(d) depicts mobile device's energy consumption when it receives data from another device through WIFI network. The received data is finally stored in disk sectors, therefore, disk encryption software will encrypt the received data if it is enabled. On average, 56% of energy consumption is cost by dm-crypt, and *Populus* can reduce it to 25%.

6.2. Evaluation on pure disk encryption/decryption operations

To compare dm-crypt with *Populus*, one effective method 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 their energy consumption cannot be directly measured by Monsoon power monitor. To overcome this



(a) Energy consumption of video playing.

(b) Energy consumption of video recording.

(c) Energy consumption of data sending through WIFI.

(d) Energy consumption of data receiving through WIFI.

Fig. 4. Energy consumption on typical usage for mobile devices.

problem, we first run a test program to conduct disk operations while using Monsoon power monitor to measure the energy cost of the whole mobile device and the time cost during the execution of the test program. Note that disk operations trigger disk encryption/decryption operations when disk encryption software is enabled so that part of the energy cost of the whole mobile device comes from disk encryption/decryption. Then we use an *energy consumption model* to estimate and compare the energy consumption of pure disk encryption/decryption operations of dm-crypt and *Populus*.

Here, we introduce the basic idea behind our model. Since our test program only triggers disk operations, the energy consumption of the whole mobile device during the execution of the test program mainly comes from three sources: disk operations, disk encryption/decryption and system maintenance. Since the energy consumption of the whole mobile device can be directly measured, we can compute the energy consumption on disk encryption/decryption if we can estimate the energy consumption on system maintenance and disk operations.

Now, we give more details on how to estimate the energy consumption on system maintenance and disk encryption. For system maintenance, we notice that the maintenance operations are conducted periodically, therefore, the energy consumption on system maintenance in a period does not change much due to the usage of disk encryption software or the execution of our test program. Hence, we first measure the energy consumption of the whole mobile device in a period without running our test program and enabling disk encryption software. Note that at this stage, the energy consumption of the whole mobile device is nearly equal to the energy consumption of system maintenance. Then we compute the average energy consumption of system maintenance per time unit as *the power of system maintenance*. To estimate the energy consumption on system maintenance during the execution of our test program, we first measure the time cost of the execution and then multiply the time cost with the power of the system maintenance as an estimated result of the energy consumption from system maintenance.

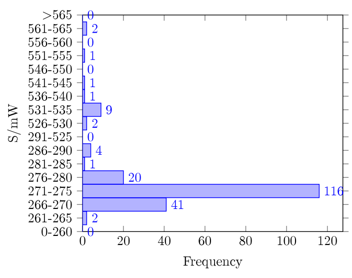
To estimate the energy consumption on disk operations, we run our test program when disk encryption software is disabled and measure the energy cost of the whole mobile device and the time cost of the execution of our test program. Note that without the disk encryption, the energy cost of the whole mobile device only comes from disk operations and system maintenance. Then we use the method discussed previously to estimate the power of system maintenance. Since the energy consumption of the whole mobile device can be directly measured and the energy consumption on system maintenance can be estimated with the time cost and the power, the energy consumption from disk operations can also be computed.

Let us formally describe our model. We define $G = \frac{A-P}{A}$, where A denotes the energy cost of dm-crypt and P denotes the energy cost of *Populus*. Our aim is to compute G, since it can reflect the percentage of the energy that *Populus* can save relative to dm-crypt. Here, we present how to estimate G with what we can directly measure. For simplicity, we define 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. Considering that the energy consumption of the whole mobile device can be separated into two (when disk encryption software is disabled) or three parts (when disk encryption software is enabled), we have $E_1 = F + T_1S$, $E_2 = F + T_2S + A$, $E_3 = F + T_3S + P$, where E_i ($i \in \{1, \ldots, 3\}$) denotes the energy consumption of the whole mobile device during the execution of the test program when the *Conf*.i is enabled, T_i ($i \in \{1, \ldots, 3\}$) denotes the time cost of the execution of the test program when the *Conf*.i is enabled, S denotes the power of system maintenance, F denotes the energy consumption of disk operations. Since E_i, T_i, S can be directly measured, we can compute G as follows:

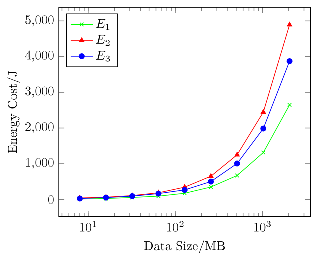
$$\begin{aligned} G &= \frac{A-P}{A} \\ &= \frac{(E_2 - F - T_2S) - (E_3 - F - T_3S)}{E_2 - F - T_2S} \\ &= \frac{(E_2 - F - T_2S) - (E_3 - F - T_3S)}{E_2 - (E_1 - T_1S) - T_2S} \\ &= \frac{(E_2 - E_3) - S(T_2 - T_3)}{(E_2 - E_1) - S(T_2 - T_1)}. \end{aligned}$$

We measure S 200 times and show the distribution of those measured results in Fig. 5(a). As one can observe, most measured results are between 261 and 290 mW while a few measured results are between 526 and 565 mW. Finally, we average all measured results and obtain 294 mW as an estimated value of S.

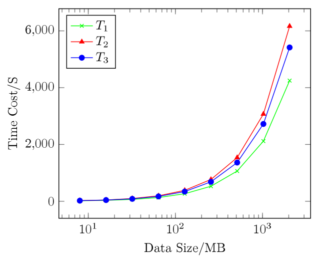
To measure E_i and T_i, we run our test program which uses JNI technique to randomly read/write files without caching data into various buffer mechanism. Fig. 5(b) shows the measured results of E_i, $i \in \{1, 2, 3\}$. When the data size (i.e., the size of data written to storage or read from storage) is small, E_1, E_2, E_3 are very close. But when the data size becomes larger, $E_2 > E_3 > E_1$ becomes more obvious. Fig. 5(c) depicts the measured results of T_i, $i \in \{1, 2, 3\}$. Similar to the trend of E_1, E_2, E_3, $T_2 > T_3 > T_1$ tends to be more obvious with the growth of data size. Fig. 5(d) depicts the estimated results of G. When the data size is small, G fluctuates between 50% and 70%. When the data size grows larger, G is close to 60%. In fact, when the data size is small, the energy consumption of disk encryption/decryption is also small so that the estimated results can be easily affected by noises in other sources



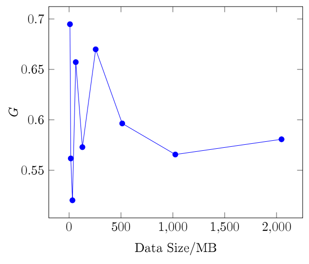
(a) The distribution of 200 samples of S.



(b) The energy cost of the whole mobile device.



(c) The time cost of our test program.



(d) The percentage of the energy that Populus saves relative to dm-crypt.

Fig. 5. Experimental results regarding the energy consumption of pure disk encryption/decryption operations.

of energy consumption. When the data size increases, the energy consumption of disk encryption/decryption becomes the dominant part. In conclusion, *Populus* saves around 50%–70% less energy than dm-crypt system.

7. Related work

Popular disk encryption software includes dm-crypt [23] (for Linux and Android), BitLocker [24] (for Windows), FileVault [25] (for Mac OS X) and etc. These systems use classic SPN-based block ciphers such as AES, BlowFish, Two Fish [26] with common disk encryption modes such as CBC and XTS [4]. These systems are secure in cryptography, but usually have a large number of CPU and RAM operations, which may cause performance overhead and energy overhead [7,27].

To solve this problem, many hardware solutions such as [28–31] were proposed to mitigate performance or energy overhead. Those hardware solutions can execute complex block ciphers in high speed without consuming much energy, but usually rely on certain hardware architecture or platform. Given compatibility and scalability, software solutions such as [5,10,32,33] were also proposed. Despite the low energy cost of those software solutions, they either have security vulnerabilities in cryptography [11] or only apply to the protection of certain data types or formats (e.g., image and video). Different from previous work, our work *Populus* is a both cryptographically secure and energy-efficient disk encryption software that can protect various types of data.

8. Conclusion

In this paper, we develop a cryptographically secure and energy-efficient disk encryption software *Populus*. *Populus* uses modular linear algebra and one-time pad technique to encrypt sensitive data. Plus, *Populus* is based on client–server pattern. Its server side helps the client side do input-free computation, thus saving energy on mobile devices. We prove *Populus* is semantically secure against state-of-the-art CPA methods. We also conduct energy consumption experiments, and our experimental results show that *Populus* consumes 50%–70% less energy when compared with dm-crypt.

Acknowledgment

The work of John C.S. Lui was supported in part by the GRF 14208816.

References

[1] M.A. Bouazzouni, E. Conchon, F. Peyrard, Trusted mobile computing: An overview of existing solutions, Future Gener. Comput. Syst. (2016).

[2] A. Lima, B. Sousa, T. Cruz, P. Simões, Security for mobile device assets: A survey, in: ICMLG2017 5th International Conference on Management Leadership and Governance, Academic Conferences and publishing limited, 2017, p. 227.

[3] T. Müller, F.C. Freiling, A systematic assessment of the security of full disk encryption, IEEE Trans. Dependable Secure Comput. 12 (5) (2015) 491–503.

[4] L. Khati, N. Mouha, D. Vergnaud, Full disk encryption: Bridging theory and practice, in: Cryptographers Track At the RSA Conference, Springer, 2017, pp. 241–257.

[5] 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.

[6] 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, Lat. Amer. Trans., IEEE (Rev. IEEE Amer. Lat.) 13 (1) (2015) 291–298.

[7] 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.

[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 Syst. 21 (1) (2015) 87–101.

[10] P. Crowley, Mercy: A fast large block cipher for disk sector encryption, in: FSE, Vol. 1978, Springer, 2000, pp. 49–63.

[11] S.R. Fluhrer, Cryptanalysis of the mercy block cipher, in: Fast Software Encryption, Springer, 2002, pp. 28–36.

[12] M. Liskov, R.L. Rivest, D. Wagner, Tweakable block ciphers, J. Cryptology 24 (3) (2011) 588–613.

[13] C.E. Shannon, Communication theory of secrecy systems*, Bell Syst. Tech. J. 28 (4) (1949) 656–715.

[14] M. Bellare, B. Tackmann, Nonce-based cryptography: Retaining security when randomness fails, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2016, pp. 729–757.

[15] P. Yadav, I. Gupta, S. Murthy, Study and analysis of estream cipher salsa and chacha, in: 2016 IEEE International Conference on Engineering and Technology, (ICETECH), IEEE, 2016, pp. 90–94.

[16] J. Sharma, D. Koppad, Low power and pipelined secure hashing algorithm-3 (SHA-3), in: India Conference (INDICON), 2016 IEEE Annual, IEEE, 2016, pp. 1–5.

[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] W. Stallings, Cryptography and Network Security, 6/E, Pearson Education, Inc., publishing as Prentice Hall, 2016.

[20] N. Sharma, S. Chirgaiya, A Novel Approach to Hill Cipher, Int. J. Comput. Appl. 108 (11) (2014) 34–37.

[21] X. Yao, Z. Chen, Y. Tian, A lightweight attribute-based encryption scheme for the Internet of Things, Future Gener. Comput. Syst. 49 (2015) 104–112.

[22] L. Zhang, Z. Yan, R. Kantola, Privacy-preserving trust management for unwanted traffic control, Future Gener. Comput. Syst. 72 (2017) 305–318.

[23] L. Demir, M. Thiery, V. Roca, J.-L. Roch, J.-M. Tenkes, Improving dm-crypt performance for xts-aes mode through extended requests: first results, in: GreHack 2016. The 4th International Symposium on Research in Grey-Hat Hacking-Aka GreHack, 2016.

[24] D. Goyal, Survey on Bitlocker Techniques, Int. J. Adv. Res. Comput. Sci. 7 (6) (2016).

[25] C.L. Baron, File vault and cloud based document notary service, Google, Patents, US Patent App. 15/138,372, (Apr. 26 2016).

[26] R. Yegireddi, R.K. Kumar, A survey on conventional encryption algorithms of cryptography, in: International Conference on, ICT in Business Industry & Government, (ICTBIG), IEEE, 2016, pp. 1–4.

[27] A. Fujimoto, P. Peterson, P. Reiher, Comparing the power of full disk encryption alternatives, in: Green Computing Conference (IGCC), 2012 International, IEEE, 2012, pp. 1–6.

[28] J. Lee, K. Ganesh, H.-J. Lee, Y. Kim, FESSD: A fast encrypted SSD employing on-chip access-control memory, IEEE Comput. Archit. Lett. (2017).

[29] D. Liu, X. Luo, Y. Li, Z. Shao, Y. Guan, An energy-efficient encryption mechanism for NVM-based main memory in mobile systems, J. Syst. Archit. 76 (2017) 47–57.

[30] M.A. Alomari, K. Samsudin, A.R. Ramli, S.J. Hashim, Efficient android-based storage encryption using multi-core cpus, Secur. Commun. Netw. 9 (18) (2016) 5673–5686.

[31] X. Luo, D. Liu, L. Liangy, Y. Li, K. Zhong, L. Long, MobiLock: An energy-aware encryption mechanism for nvram-based mobile devices, in: Non-Volatile Memory System and Applications Symposium (NVMSA), 2015 IEEE, IEEE, 2015, pp. 1–6.

[32] S. Hong, J. Im, S.M. Islam, J. You, Y. Park, Enabling Energy Efficient Image Encryption using Approximate Memoization, J. Semicond. Technol. Sci. 17 (3) (2017) 465–472.

[33] H. Xu, X. Tong, X. Meng, An efficient chaos pseudo-random number generator applied to video encryption, Optik-Int. J. Light Electron. Opt. 127 (20) (2016) 9305–9319.

Yang Hu received the B.Eng. degree in software engineering at Xi'an Jiaotong University, Xi'an, China in 2014 and received the M.Eng. degree in software engineering at Xi'an Jiaotong University in 2017. He was a research assistant in the CSE Department at The Chinese University of Hong Kong, China. His research interests include network/system security, machine learning and data mining.

John C.S. Lui is currently the Choh-Ming Li Chair Professor in the CSE Department at The Chinese University of Hong Kong. His current research interests are in Internet, network sciences with large data implications, machine learning on large data analytics, network/system security, network economics, large scale distributed systems and performance evaluation theory. He received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award. John also received the CUHK Faculty of Engineering Research Excellence Award (2011–2012). John is a co-recipient of the best paper award in the IFIP WG 7.3 Performance 2005, IEEE/IFIP NOMS. He is a Fellow of ACM and IEEE.

Wenjun Hu received the B.S. degree and the M.Sc. degree in control science and engineering at Xi'an Jiaotong University, Xi'an, China. He is currently a malware research engineer at Palo Alto Networks, Inc. His research interests include behavior analysis of Android applications and Android malware detection.

Xiaobo Ma received the Ph.D. degree in control science and engineering from Xi'an Jiaotong University, Xi'an, China in 2014. He was a Post-Doctoral Fellow with the department of computing, Hong Kong Polytechnic University from Feb. 2015 to Feb. 2016. He is currently an assistant professor with the department of computer science and technology of Xi'an Jiaotong University, as well as a research faculty with the MOE KLINNS Lab. His research interests include network security and privacy, vehicle network, and mobile systems.

Jianfeng Li received the B.S. degree in automation engineering in 2010 from Xi'an Jiaotong University, Xi'an, China. He is currently Ph.D. candidate with the System Engineering Institute and MOE KLINNS Lab of Xi'an Jiaotong University. His research interests include network modeling, measurement and botnet detection.

Xiao Liang received the B.Eng. degree in software engineering at Xi'an Jiaotong University in 2014. He received the M.Sc. degree in control science and engineering at Xi'an Jiaotong University in 2017. His research interests include mobile system security, natural language processing, machine learning and deep learning.