2015 IEEE Symposium on Service-Oriented System Engineering

A Lightweight Encryption and Secure Protocol for Smartphone Cloud[1]

William Zegers
Computer Engineering
San Jose State University
San Jose, USA
william.zegers@sjsu.edu

Sang-Yoon Chang
Advanced Digital Science Center
Singapore, Singapore
sychg@adsc.com.sg

Younghee Park[2], Jerry Gao
Computer Engineering
San Jose State University
San Jose, USA
{younghee.park, jerry.gao}@sjsu.edu

Abstract— **User data on mobile devices are always transferred into Cloud for flexible and location-independent access to services and resources. The issues of data security and privacy data have been often reverted to contractual partners and trusted third parties. As a matter of fact, to project data, data encryption and user authentication are fundamental requirements between the mobile devices and the Cloud before a data transfer. However, due to limited resources of the smartphones and the unawareness of security from users, data encryption has been the last priority in mobile devices, and the authentication between two entities always depends on a trusted third party. In this paper, we propose a lightweight encryption algorithm and a security handshaking protocol for use specifically between in mobile devices and in Cloud, with the intent of securing data on the user side before it is migrated to cloud storages. The proposed cryptographic scheme and security protocol make use of unique device specific identifiers and user supplied credentials. It aims to achieve a users-oriented approach for Smartphone Cloud. Through experiments, we demonstrated that the proposed cryptographic scheme requires less power consumption on mobile devices.**

Keywords; Security, Mobile devices and smartphones, Android, Cryptography, Cloud

I. INTRODUCTION

Over the past decade, mobile devices such as smart phones and tablets have caused a paradigm shift into nearly every field of the computing industry, including wireless networking, web-based business models, and the methods by which data are transmitted between clients, servers, or other clients. Furthermore, this shift has meant services and data becoming more readily available to mobile device users; consumers no longer require access to traditional, more stationary computers to be able to access web-based services, communicate, or access personal or shared data.

Recently, customer data on a device is automatically backed up to clouds as clouding environment has been popularly deployed like iCloud, Azure, EC2, and so on. To protect the customer data, data encryption is an unavoidable solution. However, mobile devices have a pain to process the data encryption due to limited power and resources even though many data encryption algorithms have been proposed for several decades. Usually, encryption has been diminished in order to improve processing speed [6, 11, 12].

Since version 4.0, Android has included the ability for whole disk encryption [4], many mobile solutions exist for users to encrypt data and individual file locally, or to encrypt data to by stored in the cloud. However, these tend to use methods originally designed for systems where resources such as battery life and memory footprint are less constrained than mobile devices. For large exchanges of data between mobile devices and remote storage servers, the additional overhead incurred by encryption and decryption can accumulate to be detrimental to the resources of mobile devices. To mitigate the threat of data theft, it is evident that a reliable encryption scheme is needed to protect data – both physically and while in transit to cloud services – while taking into account the resource constraints of battery-powered mobile devices.

In this paper, a lightweight encryption scheme is proposed on a block cipher. It consists of substitutions, permutations, and rotations. The key is securely generated by a user credential and unique device information. The proposed method aims to minimize the amount of processing time and power consumption, comparing to existing mobile encryption applications. Furthermore, a remote secure authentication protocol is proposed without a third party authority. The security protocol utilizes a user password and device information to authenticate the user and the cloud service provider. Each party can generate the common secret key based on its own information. The proposed security protocol contributes to the minimum communication overhead since they just need to exchange random numbers. The proposed methods aim to achieve user-centric encryption and authentication instead of depending on a third party.

This paper introduces a new scheme to encrypt data on the device side that uses the native APIs of the Android framework. Implemented specifically for use on the Android mobile platform, the scheme is designed to use the functionality and APIs provided by the Android framework itself to provide encryption with the goal that by catering specifically to a mobile-based platform, resources use is more tailored for mobile devices and able to reduce the overhead caused by traditional encryption schemes. This

[1]This research was supported by SJSU undergraduate research award and in part by Singapore's Agency for Science, Technology and Research (A*STAR), under a research grant for the Human-centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center.
[2]This author is a corresponding author.

reduced resource use, in turn, will be more suited to allow Android devices to exchange and encrypt/decrypt larger volumes of data with cloud-storage servers. Furthermore, our scheme uses principles of secure encryption, such as non-linearity, cipher block chaining, and byte scrambling, to protect confidentiality of data.

This paper contributes to the development of the new lightweight encryption algorithm and the new security protocol for mutual authentications for mobile cloud computing. We develop a usable encryption application for Android phones and evaluate it on a real smartphone device. Through experiments, the proposed method demonstrates the feasibility of user-centric encryption and authentication scheme for mobile cloud computing.

In section II of this paper, we outline a proposed method for the design of this encryption scheme, describing the algorithm, scheme, and implementation of the encryption program itself as an Android application. Section III compares the proposed method with other methods currently in use in the industry. The experimental design to measure the performance of the proposed scheme is described in section IV, detailing the setup and measurements taken to determine how the proposed method performs. Section V examines the results of section IV, namely by examining the implications of these results and how they compare to other available encryption schemes. Finally, section VI concludes the paper with an overview of the previous sections.

II. The Proposed Method

A new encryption algorithm and secure handshaking protocol are proposed to protect data on mobile phones based on a user password and a unique device identifier. It is based on a user-centric method instead of depending Cloud Service Providers (CSP). It aims to encrypt personal data on the limited resource environment before a data transfer into Cloud. When users need to access their data, authentication is performed between the smartphones and the CSP based on the device information. The following sections will be given a description of the proposed methods in detail.

A. A Lightweight Encryption Algorithm

Encryption consists of an initial XOR operation, followed by several steps of matrix transformations to shuffle the data; decryption is simply reverse of the encryption procedure, using inverted lookup tables for the substitution and column permutation steps. Each block of data undergoes seven rounds of encryption (or decryption), and the scheme uses CBC mode to minimize the chance of two blocks of plaintext producing identical blocks of ciphertext.

Fig. 1 gives the overall architecture of the round-based encryption scheme. It is designed as a block cipher using 144-bit keys that operates on 128-bit blocks at a time. The encryption procedure is repeated for a total of seven rounds, using a unique 144-bit key for each round. Fig. 2 provides a precise algorithm converts each plaintext block to a cipher text block. Based on Fig. 2, each components are described in detail.

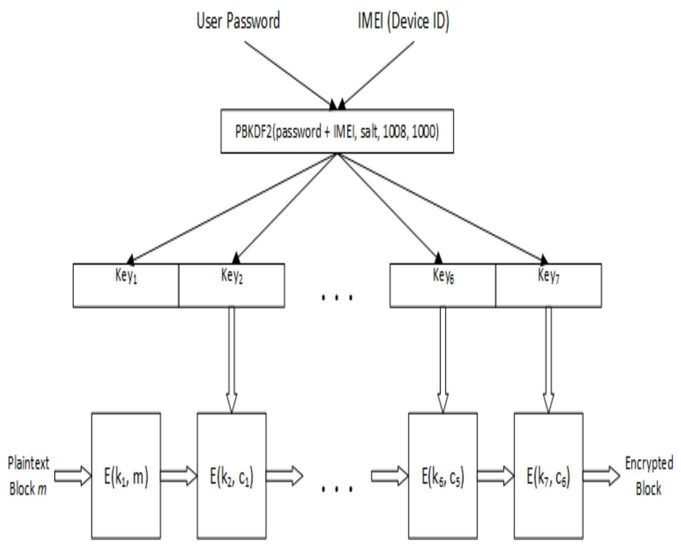


Figure 1. The round-based encryption scheme used to encrypt each block of data, consisting of seven rounds, with each round using a unique round and permutation key, to produce an encrypted ciphertext block.

The architecture uses seven repeated rounds, each with its own independent key, to ensure thorough obfuscation of the orginal data while still remaining lightweight. So while each block is being encrypted by running it through several rounds of the encryption algorithm, the architecture uses as few rounds as possible to limit consuming too much processing power.

The key is derived from a user-supplied password and device ID, using a unique hardware identification with something only the authorized knows to generate the 1008-bit key. By using two sources of input to derive the key, this helps to limit that opportunity to obtain full credentials to decrypt the data.

1) Key Derivation: The derived key is, consequently, based on something the user knows (e.g. a password) and something the user has (i.e. a unique device identifier), making decryption of the encrypted data possible only by the user who knows the password and on the device on which it was originally encrypted. Only authorized user can generate a right key based on the two factors that can be selected from various data, such as user biometric information, device unique numbers (e.g. IMEI, MEID, ESN, IMSI based on Android) [14]. We use PBKDF2 (Password-based Key Derivation Function) [15] as shown in Fig. 1 and it generates a 1008-bit key from a user password and IMEI(device ID) that we choose.

The derived key is used as round, XOR, and permutation keys after breakup. In other words, the 1008-bit key is broken into seven 144-bit round keys, each of which is further broken down into two parts: a 128-bit XOR key and a 16-bit permutation key that determines the permutations of the block in later steps of the algorithm.

2) Cipher Block Chaining(CBC): Before being fed to the into the actual encryption algorithm, the scheme uses CBC mode to change the bytes of the plaintext data; that is, each block of plaintext is XORed with the previous block of ciphertext. The result of the XOR then becomes the input for the encryption function. In the case of the first block, where no previous block exists, the salt used to seed the password is also used for a random IV to XOR with the first block of data. Each 128-bit plaintext block is XORed with the 128-bit round key to produce a new block consisting of ciphertext.

3) Substitution: Using Rijndael substitution blocks [16], each byte in the XORed block is substituted for a new byte using a basic lookup table. This further reduces the linearity of the algorithms, making it less susceptible to differential and linear cryptanalysis. The substitution is performed by a one-dimensional array constructed from the Rijndael S-boxe in [16]. After substitution, the newly substituted bytes in the ciphertext block are, from this point forward, treated as a 4 x 4 matrix. The matrix after configuring blocks is processed by the step of column permutation.

	00	01	10	11
00xx	1243	2341	3412	1342
01xx	4231	2314	4312	3421
10xx	4132	4123	1234	2413
11xx	3124	1423	3241	2134

Table 1. Column permutation done using 4 bits from the permutation key, with each cell showing the re-ordering of the original column order.

4) Column permutation: The column permutation is performed two times before and after *Row Rotation* as shown in Figure 2. In the first column permutation, the first four bits of the 16 bit permutation key determine how the columns of the 4 x 4 matrix are permuted using a lookup table provided in the code of the application in Table 1. Given an input block, a 4-bit column permutation index, and a one-dimensional array constructed from Table 1. In the second column permutation, the procedure permutes the columns once more to shuffle the ciphertext bytes using the last four bits of the permutation key.

5) Row rotation: The next 8-bit are split amongst the 4 rows, 2 per row, and determine the amount by which the row gets rotated to the left. For example, a bit series of 10 would rotate the row two to the left, while a 00 bit series would cause the row to remain the same. Since the four 2-bit rotation number sit over the border of two bytes, we use individual bit manipulation to extract these numbers and apply them to their corresponding row.

6) Padding: In the case that the final block of the data to be encrypted is not a full of 16 bytes, additional bytes are added to make the same block size. Block padding is handled when each block is being read from the file to be encrypted. If the method reaches the end of file marker, it checks the number of bytes read into the current block and, if this number is less than sixteen bytes, it pads the block with the appropriate bytes and continues to encrypt the block as usual. For example, if the final block is only 10 bytes long, six copies of the byte `0x06` are appended to the block to create sixteen bytes. During decryption, removal of these bytes is done by simply examining the last byte of the last block, and removing that many bytes.

The encryption procedure is repeated for a total of seven rounds, using a unique 144-bit keys for each round. The proposed cryptography system is designed by a typical block cipher based on substitutions and permutations. The key is generated by a user password and a unique device identification.

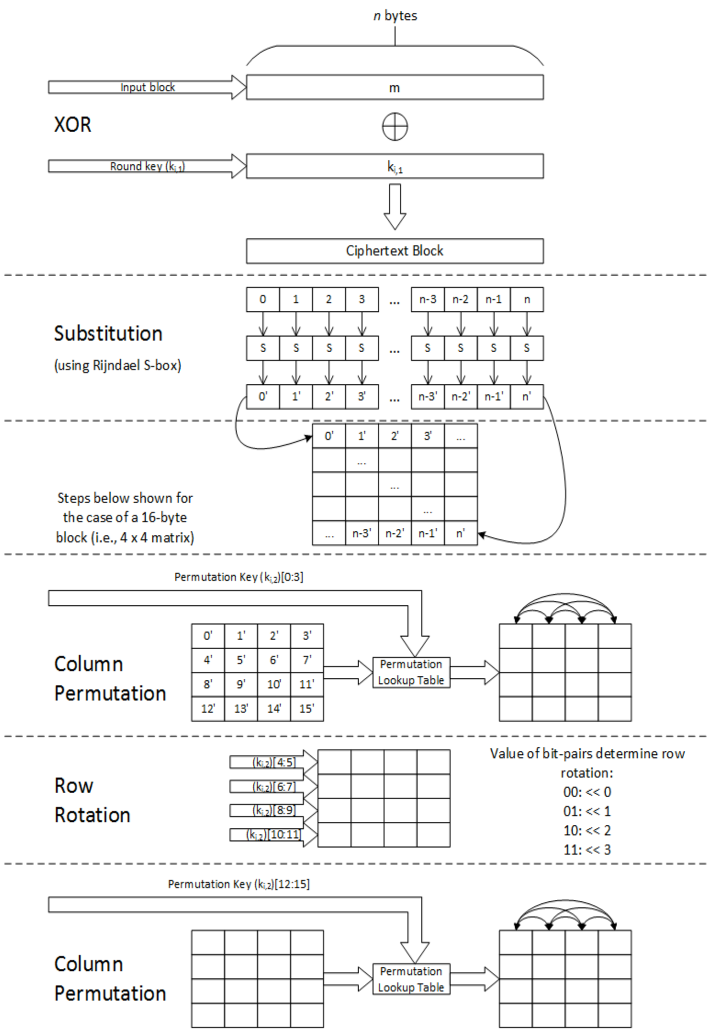


Figure 2. A diagram of the encryption algorithm being implemented, showing the steps and function carried out on each 128-bit data block.

B. *Decryption Algorithm*

Decryption algorithm is described according to the encryption scheme in reverse order. First, we generate the same key based on the PBKDF2 function with the same user password and the device's IMEI in order to decrypt the encrypted data. [SYC: Referring back to the Key Derivation Section to remind the readers how the receiver (whether it is the transmitter itself or not) is capable of key generation and decryption will be helpful.] Second, now the decryption process is similar to the encryption method as follows.

1) Permute the columns using the last four bits of the permutation key to look up the specific permutation.
2) Use permutation key bits 4 through 11 to rotate the rows to the *right*, similar to the fashion described in step 7 of encryption.
3) Permute the columns again using the first four bits of the permutation key.
4) Substitute each byte of the block using the *inverse* Rijndael S-box.
5) XOR the bytes with the round key.
6) Repeat steps 1 through 5 for seven rounds to generate the original plaintext of the data.
7) Finally, the decryption algorithm undoes the XOR operation from CBC mode by outputting the XOR of the previous ciphertext with the output of steps 1 – 6 to reproduce the initial plaintext.

	00	01	10	11
00xx	1243	4123	3412	1423
01xx	4231	3124	3421	4312
10xx	2431	2341	1234	3142
11xx	2314	1342	4213	2134

Figure 3. Inverse column permutations used during decryption

Both the column permutations and the Rijndael S-boxes from the encryption method are inverted revert the data back to its original positions. Figure 3 gives the inverted column permutations and the reverse Rijndael S-box is also used during the substitution step of decryption, respectively. After a successful decryption, the method removes the padding bytes from the end of the decrypted file. The result is a decrypted copy of the encrypted file.

C. *User Authetication*

This paper proposes a remote secure user authentication protocol in order to allow users to access their data without disclosing any secret information over communications between users and the cloud service provider (CSP). Section II.A briefly mentioned the scenario of implementing the proposed encryption on a mobile cloud service provider, allowing for more universal access of the data. However, this means that at some point, devices using the mobile cloud would need to exchange the IMEI, provided password in order for the provider to be able to authenticate users. This could, of course, be accomplished at the initial stage wherein a user is registering her device with the mobile cloud provider and, using an agreed upon the use of cloud service.

Fig. 3 shows a user authentication protocol based on a user password, a device identification number, and random numbers. As shown in Fig. 3, the authorized users and CSP can compute a shared secret key K, where K is $g^{b(a+uW)}$ *mod* p. Note that $\boldsymbol{p}$ is a large prime number known to both entities, $\boldsymbol{g}$ is a generator in the finite field $\boldsymbol{Fp}$, and $\boldsymbol{a}$ and $\boldsymbol{b}$ is a random number only known to each entity. The CSP stores g^W *mod* p, where $\boldsymbol{W}$ is a function of the user password, instead of saving the user password. The user can also calculate $\boldsymbol{W}$ from his or her own password. Both of them can also generate $\boldsymbol{u}$, where $\boldsymbol{u}$ is a 32-bit number derived from the user device identification number (e.g. IMEI, MEID, ESN, IMSI based on Android).

Based on the initial settings, the CSP can compute $g^{b(a+uW)}$ *mod* p since it knows $\boldsymbol{b}$, $\boldsymbol{W}$, $\boldsymbol{u}$. The mobile user can also compute the same key K because he knows $\boldsymbol{a}$, $\boldsymbol{W}$, $\boldsymbol{u}$. Based on the generated shared key, the last two steps can finish to authenticate each other with a nonce each time, as described in Fig. 3. During the communications, any secret information is never exchanged. Even though attackers intercept any conversations, they cannot generate the temporary session keys or user passwords. The nonce, $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ can be changed with a timestamp to avoid replay attacks in the future.

The benefit of the proposed protocol does not rely on a third-party authority (i.e. Certificate Authority, CA) to negotiate a connection between the device and mobile cloud. The third-party authority causes a lot of challenging issues: a deployment, a single-point failure, a certificate management, revocation, and so on [30]. However, the proposed authentication protocol is performed between two parties without a third-part authority. It is based on two factors by using a password and unique device information for mutual authentication.

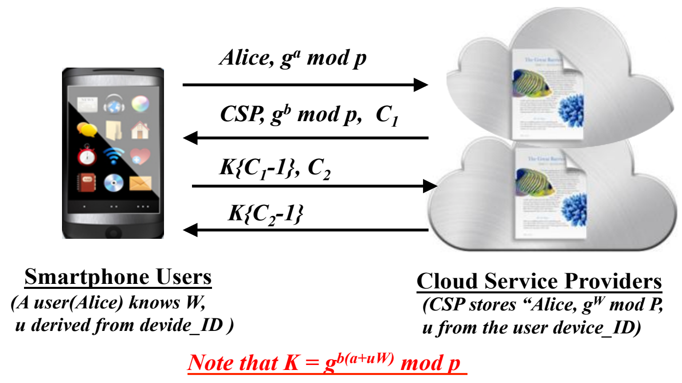


Figure 3. A secure remote password protocol for user authentication. The CSP has g^W *mod p*, where W is a function of the user password. The user can calculate W from his or her own password.

III. Evaluation

A. Implmentation and Experiment in Software

To evaluate the proposed method, it was implemented on Java platform SE 7 with the Android SDK to create an application. In addition, to generate a key, we used Password-based Key Derivation Function 2 (PBKDF2) from a user password concatenated with the device IMEI as the seed. It is a standard cryptographic function that takes a seed, salt, generated key size, iteration count, and hashing algorithm to produce the encryption keys. For our method, the algorithm generates a key with size 1008 bits (i.e., 144 bits times seven rounds). The salt and iteration count are used to slow any offline attacks on a password-based key derivation function, such as dictionary and brute force attacks. The salt constitutes the header of the encrypted file so it can be easily retrieved during decryption, which is then followed by the body of encrypted bytes.

To compare the performance of the proposed method with other available methods, two popular encryption applications: Cryptonite and DroidCrypt. We used our device to evaluate performance overhead for the proposed method. We used Nexus 7 with Android 4.4.2, 1GB RAM, Non-removable Li-Ion 4325 mAh battery (16 Wh). Qualcomm Trepn Profiler[1] is used to experiment power consumption and CPU frequency for different available encryption applications for Android. Trepn Profiler is a diagnostic tool to profile the performance and power consumption of Android applications running on devices.

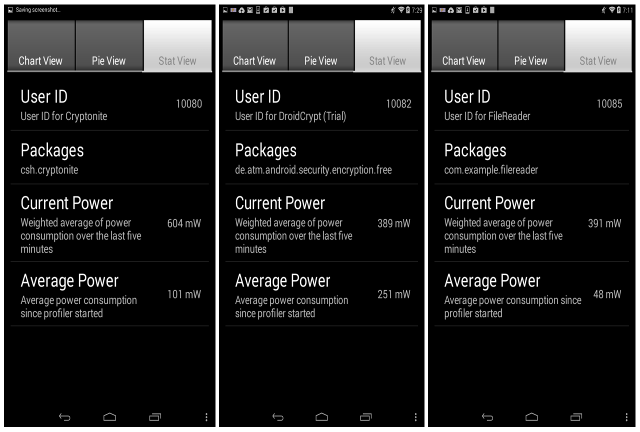
Table 1 shows the summary of each application for our evaluation. We selected two different applications. The two applications are based on Advanced Encryption Standard (AES), one of block ciphers. To compare our proposed algorithm, we use the same key size and block size as shown in Table 1. However, the applications also support different algorithms like Blowfish, RC6.

Table 1. Cryptographic applications for Android devices.

	Algorithms	Key Size	Block Size
Crytonite	AES	128	128
DroidCrypt	AES	128	128
Our Approach	-	144	128

Fig. 4 demonstrated our application keep maintaining the smallest power when we encrypted 100KB file. The current power usage indicated the total power consumptions after starting each application on the device. The average power usage indicated the power consumption on average after the Trepn Profiler started. The CPU frequency was 1.026 MHz for our approach. The other application showed 1.512MHz CPU frequency to encrypt 100KB file.

[1]QUALCOMM Trepn Profiler at https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler.



(a)

User ID — 10080
User ID for Cryptonite

Packages
csh.cryptonite

Current Power — 604 mW
Weighted average of power consumption over the last five minutes

Average Power — 101 mW
Average power consumption since profiler started

(b)

User ID — 10082
User ID for DroidCrypt (Trial)

Packages
de.atm.android.security.encryption.free

Current Power — 389 mW
Weighted average of power consumption over the last five minutes

Average Power — 251 mW
Average power consumption since profiler started

(c)

User ID — 10085
User ID for FileReader

Packages
com.example.filereader

Current Power — 391 mW
Weighted average of power consumption over the last five minutes

Average Power — 48 mW
Average power consumption since profiler started

(a) Cryptonite (b) DroidCrypt (c) Our Approach

Figure 4. Evaluation for power consumption for Android applications.

B. Implmentation and Experiment in Hardware

In order to examine the performance of the algorithm and regards to processing resources consumed and speed, the proposed algorithm was also simulated at the hardware level using Verilog HDL. The PBKDF was ignored for the hardware implementation to focus on the performance of the algorithm itself, so the simulation supplied a hard-coded key to the algorithm without any generation.

To judge performance, we examined the execution time of the proposed algorithm and compared it to the execution of the Data Encryption Standard (DES), similar to AES in a block cipher. Setup times of signals between registers were examined, and the longest signal setup time for each algorithm was deemed to be the limiting factor of the clock-cyle time; similar to how a manufacturing assembly line can only output a product as fast as the slowest stage in line. The limiting cycle-time was multiplied by the total number of cycles necessary to encrypt one 16-byte block of data to give the total execution time per block of data; that is:

$$t_{execution} = t_{slowest\ instr.}\ X\ cycles/block$$

Timing analysis of the hardware modules, the basis for estimating the cycle-time per instruction, after placement and routing of the HDL code. This analysis may be in slight disagreement with actual results were the algorithms implemented on actual hardware, but the estimation was deemed suitable to compare two algorithms.

Hardware testing in Verilog gave very promising results, though problems with some module implementation would not allow for a thorough analysis of all signals due to limitations of the simulation software.

The DES was given as 12.930 *ns* inside the module that handled the initial permutation of data encryption. Furthermore, the total number of cycles necessary to encrypt one block of data was found to be a total instruction count of 81 instructions, giving a total execution time per block encryption of about 1.050 μs.

For our proposed algorithm, obtaining signal setup times proved to be slightly more challenging, as some of the modules could not be full simulated as hardware by the simulation software. Nonetheless, we obtained a maximum cycle time of 7.482 *ns* and a total instruction count of 21 instructions. This execution time of our result when implemented in hardware turned out to be a significant improvement over DES, giving a total of 0.160 μs execution time per block of encrypted data.

IV. Discussion

Our motivation in the design of our encryption scheme was to build a lightweight and low-power method that could be implemented on mobile devices, with a particular focus on securing data to be stored in the cloud. On the cloud – and even during transmission from device to cloud – the data exists outside of the user's immediate control, and how effectively it is secured is dependent on the storage provider's security. This security could potentially have many risks that a user may deem unacceptable to host unencrypted data: a security breach by cyber criminals, a malicious or simply careless administrator, or even user error when dealing with a new or unfamiliar system.

The encryption scheme is intended to encrypt data on the user's device before it is transmitted and stored in cloud storage. Moreover, since it is being run on a mobile device where power consumption and memory usage are much more constrained than on traditional computers, we designed the algorithm with the goal of having a reduced overall CPU usage and memory footprint versus other encryption methods.

Of course, encrypting only a few kilobytes or even megabytes of data at a time on a mobile device will show little impact on the amount of power and memory being consumed. However, as device users are migrating their data from local to cloud storage, sending and receiving data that must encrypted and decrypted, ever-growing exchanges of data will put larger and larger demands on resource use. A high enough volume of data exchange that needs encryption and decryption to ensure its confidentiality can accumulate to become a significant enough drain on limited system resources.

With this in mind, our algorithm was designed to be simple and efficient. We constructed the algorithm to be easily implemented on the Android system, using a simple XOR operation followed by several rounds of key-based matrix transformations as the backbone of the scheme. With our base encryption algorithm, we added further modifications and steps to it in order to ensure such security and cryptographic principles as non-linearity, by using byte substitution, and semantic security, using multiple rounds of encryption and running the entire scheme in CBC mode, were included in our encryption scheme.

The method operates at the byte level of data, so it is able to encrypt any file regardless of type or size. Encryption of the file does not cause any significant expansion in file size, only an additional 32 bytes for the prepended salt and "encrypted" string, plus a maximum of 16 bytes resulting from padding the final block. For the average file likely to be encrypted, these additional bytes are negligible.

We believe our algorithm is a step in building a bridge between increased demands, both personal and business, for mobile security, while at the same time being able to make safe, effective use of cloud-storage systems without having a detrimental impact to resources and performance on mobile systems.

V. Releated Work

Rapid mobile device proliferation and the maturing of cloud computing technologies introduces new challenges and opportunities in the field of data security and privacy, as work continues to boost mobile device storage and processing resources offered by cloud computing. Security architectures and encryption schemes for mobile cloud computing have been proposed by others working to find an effective way to secure data stored on an external, remote site while at the same time limiting the amount of resources, such as memory footprint and CPU usage, such security measures would incur.

A. Symmetric and Asymmetric Encryption

There exists much debate over whether using symmetric or asymmetric encryption schemes are best for data encryption. Symmetric encryption allows for faster encryption of data at a relatively low resource cost; asymmetric, on the other\hand, is much slower and requires more resources to attain the same level of security as symmetric, but does not have the problems often associated with key management as symmetric encryption.

Subasree and Sakthivel introduce in [17] a protocol using ECC to encrypt data, as well as Dual RSA to encrypt the message hash used to ensure integrity. Since this protocol relies on ECC and RSA, both of which are asymmetric encryption schemes, it is significantly slower than symmetric encryption.

Alternatively, hybrid schemes that include both symmetric and asymmetric elements have been proposed in [18], [19], and [20]. Kader et al not only evaluate several of these proposed protocols, including those mentioned in [19] and [20], but also propose their own New Hybrid Encryption Protocol (NHEP). In this protocol, block of data are divided in half, encrypting the first half using AES (symmetric) with ECC (asymmetric), and the second half using Blowfish (symmetric) with RSA (asymmetric). For both halves, the respective asymmetric encryptions are used to encrypt the secret key, which is used by the symmetric algorithms to encrypt the data. Experimental testing of

NHEP showed a significant edge over other hybrid encryption algorithms.

Often in symmetric encryption, a difficulty with secret management comes from the recommendations on key lifetimes and updates, after which a key can no longer be considered secure and data encrypted under this key must be re-encrypted using a new key. The work of Watanabe and Yoshino in [21] present a way to mitigate the risk of "key leakage" during the periodic key update. In their proposed scheme, the authors use symmetric encryption with an all-or-nothing transform add an increased level of security to encrypted data. In this scheme, the all-or-nothing transform ensures that even if part of the secret key and ciphertext is leaked, without the whole key and ciphertext, any obtained information from this leakage reveals nothing about the plaintext and is, effectively, garbage data.

B. Full Disk Encryption

One option for protecting data on mobile devices is to use a full disk encryption solution, wherein the Android operating system initially encrypts existing data on an entire storage volume, as well as all bytes written to it thereafter. On devices running the Android operating system, full disk encryption has been available since version 3.0 (aka, Honeycomb). Disk encryption on Android is handled throughout API calls to the kernel's crypto library; however, at the application level, the process is a fairly intuitive series of steps that guides the user through the encryption process. While it can take up to an hour to encrypt a device, the ease of use makes full-disk encryption an attractive option for protecting *all* data on their phone.

Wang et al also developed an alternative file system-based encryption scheme in [13] using Filesystem in Userspace (FUSE) kernel support, EncFS binaries, and *libfuse*. While the others claim the installation of their proposed encrypted file system is easy to install, it does require both rooting an Android device and flashing with a customized kernel, making adoption by a large market unlikely. The authors noted a significant cost in overhead and slowdown in database transactions using an encrypted file systems, but were also able to optimize it for 10% to 60% performance gains.

While full disk encryption methods are generally easy to use and implement on a device, they also suffer from several drawbacks. As noted in [13], encrypting an entire partition can be noticeably detrimental to system performance and resource usage and may, in the case of larger disks, require a significant amount of setup time. Moreover, full disk encryption will encrypt all data on a device – even unused or garbage bytes – and does not provide the user fine-tune controls over what is to be encrypted.

C. Identity-based Cryptography and Biometrics

Our proposed scheme uses host-based parameter of the device identy (specifically, the mobile phone's unique IMEI) to derive the cryptographic key. Identity-based cryptography, first proposed by Shamir [23], is widely used in publicly accessible environment, such as WiFi, and offers secure (symmetric) session key exchange protocols leveraging public-private key cryptography (where the identity information is used as public keys) [24, 25, 26]. In contrast to the prior work, however, we focus on physically small and computationally constrained mobile phone devices (as opposed to more capable machines such as computers), and our contribution focus is on reducing the processing load of the key generation and the encryption process. Furthermore, we use password in addition to the device identity to increase the key entropy.

Biometrics has also been used as user-unique credentials for cryptography, as is evident with the emergence of consumer-grade phones using body contact and biometrics for authentication, e.g., Apple's iPhone 6. Of particular interest is work done by Zhao et al in [20] in deploying biometric encryption (BE), using something a person *is* to either control access to an encryption key, generate an encryption key, or is bound directly to the key itself. Their work examines the possibility of using BE in mobile cloud computing, using an underlying framework that uses biometric data to handle data encryption. While BE is can potentially be a user-friendly and very secure way associating a physical characteristic to an encryption key, BE still faces many problems, such as a high false negative rate, sensor subversion, and high power consumption, until it can be widely deployed. Further cryptographic applications of biometrics are in wearable or implantable devices [27, 28, 29]. These body-touching devices share many similar traits as mobile phones in that they support mobility, are increasingly becoming intelligent with growing computing capabilities, and are constrained with the limited processing resources in both size and power. These application domains have high synergy with our research principles of lightweight computing but are beyond the scope of this paper. Thus, we leave the use of biometrics and other user credentials for security as future work.

VI. CONCLUSION

This paper proposed a lightweight encryption algorithm designed on mobile devices, with the goal of reducing the power consumption and resource usage. By doing so, it can a practical solution for encrypting large volumes of data in our local mobile devices to preserve data security. It enables end users to protect their data themselves instead of fully trusting the Cloud vendors. The experimental results of the proposed method show encouraging performance compared to other Android encryption applications, and could be a strong approach to being able to move large volumes of encrypted data to and from cloud storage.

Our current procedure relies on two unique data to seed the key that is used in the encryption algorithm: a user password and the device ID. In other words, this key is seeded by something the user *knows* (the password) and something the user *has* (the physical device on which the ID is stored). Continuing in this manner, the seed could include a third item – something the user *is* (i.e., biometrics) – to add an additional element of security to key derivation and encryption. This approach allows only legitimate users to generate their keys with a small effort to remember what users knows or has.

Based on the unique information for mobile devices, this paper also proposes a secure remote user authentication protocol without exchanging sensitive information. It doesn't require a third-party authority for mutual authentication. It can be performed by minimal communication overhead.

We would also like to further explore ways to optimize the algorithm in terms of security, easy-of-use, performance and scalability. We are currently investigating the impact certain steps of the algorithm play in the performance and resource use of procedure, as well as what role they play in strengthening the security of the algorithm. Additionally, we will consider using a more parallelizable mode of encryption over cipher block chaining such as counter mode (CTR) or CBC with ciphertext stealing (CTS).

References

[1] A. Braga, "Integrated Technologies for Communication Security on Mobile Devices," MOBILITY 2013, The Third International Conference on Mobile Services, Resources, and Users, November 2013.

[2] B. Dominic, and R. Crina, "Steganography and Cryptography on Mobile Platforms," Constanta Maritime University Annals 20, 2013.

[3] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," Advances in Cryptology—CRYPTO'99, January 1999.

[4] J. Götzfried and T. Müller, "Analysing Android's Full Disk Encryption Feature," Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), (2014)

[5] R. Jain, R. Jejurkar, S. Chopade, S. Vaidya, and M. Sanap, "AES Algorithm Using 512 Bit Key Implementation for Secure Communication."

[6] B. Bhargava, C. Shi, and S. Wang, "MPEG video encryption algorithms." Multimedia Tools and Applications, 2004.

[7] R. K. Y. Nishika, "A Lookup Table Based Secure Cryptographic SMS Communication on Android Environment," 2013.

[8] A. Patil and R. Goudar, "Sensitive Data Storage in Wireless Devices Using AES Algorithm."

[9] A. Skillen, D. Barrera,and P. C. van Oorschot, (2013, November). "Deadbolt: locking down android disk encryption." Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices, November 2013.

[10] G. J. Simmons, "Symmetric and asymmetric encryption," ACM Computing Surveys (CSUR), 1979.

[11] Z. Liu, D. Peng, Y. Zheng, and J. Liu, "Communication protection in IP-based video surveillance system," IEEE International Symposium on Multimedia, December 2005.

[12] L. Qiao and K. Nahrstedt, "Comparison of MPEG encryption algorithms," In Internation Journal on Computer & Graphics, 1998.

[13] Z. Wang, R. Murmuria, and A. Stavrou, "Implementing and optimizing an encryption filesystem on android," Mobile Data Management (MDM), July 2012.

[14] Android Developer Guide, The Developer's Guide-Android, https://developer.android.com.

[15] B. Kaliski, "Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[16] Joan Daemen and Vincent Rijmen, "The Design of Rijndael, AES - The Advanced Encryption Standard, " Springer-Verlag 2002.

[17] S. Subasree annd N. K. Sakthivel, "Design of a new security protocol using hybrid cryptography algorithms," IJRRAS, February 2, 2010.

[18] H. M. A. Kader, M. M. Hadhoud, S. M. El-Sayed, and D. S. AbdElminaam, "Performance Evaluation Of New Hybrid Encryption Algorithms To Be Used For Mobile Cloud Computing," 2014.

[19] N. Kumar, "A Secure communication wireless sensor network through hybrid (AES+ECC) algorithm", von LAP LAMBERT Academic Publishing, 2012.

[20] W. Ren and Z. Miao, "A new security protocol using hybrid cryptography," International Computer Engineering Conference (ICENCO), 2013.

[21] D. Watanabe and M. Yoshino. "Key Update Mechanism for Network Storage of Encrypted Data." Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference, 2013.

[22] K. Zhao, H. Jin, D. Zou, G. Chen, and W. Dai. "Feasibility of Deploying Biometric Encryption in Mobile Cloud Computing." ChinaGrid Annual Conference (ChinaGrid), 2013.

[23] A. Shamir, "Identity-based Cryptosystems and Signature Schemes," Lecutre Notes in Computer Science, vol. 196, pp. 47-53, 1985.

[24] J. Choi, S. Y. Chang, D. Ko, Y. Ch. Hu, "Secure MAC-Layer Protocol for Captive Portals in Wireless Hotspots," IEEE Internation Conference on Communications (ICC), 2011.

[25] S. Taha and X. Shen. "A Link-Laeyer Authentication and Key Agreement Scheme for Mobile Public Hotspots in NEMO based VANET," IEEE Global Communications Conference (Globecom), 2012.

[26] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public Key Encryption with Keyword Search," Internation Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), 2004.

[27] M. Rostami, W. Burleson, A. Juels, and F. Koushanfar, "Balancing Security and Utility in Medical Devices?" Design Automation Conference (DAC), 2013.

[28] M. Rushanan, A. D. Rubin, D. F. Kune, and C. M. Swanson, "SoK: Security and Privacy in Medical Devices and Body Area Network," IEEE Symposium on Security & Privacy (S&P), 2014.

[29] S. Y. Chang, Y. C. Hu, H. Anderson, T. Fu, and E. Huang, "Body Area Network Security: Robust Key Establishment Using Human Body Channel," USENIX Workshop on Health Security and Privacy (HealthSec), 2012.

[30] Syam Kumar P, Subramanian R, "An Efficient and Secure Protocol for Ensuring Data Storage Security in Cloud Computing," International Journal of Computer Science, Vol. 8, Issue 6, No 1, November 2011.