# FEATHER: A Proposed Lightweight Protocol for Mobile Cloud Computing Security

Ahmed Alamer
Department of Computer Science and Information Technology, School of Engineering and Mathematical Sciences, La Trobe University, Australia and Department of Mathematics, Tabuk University, Saudi Arabia
a.alamer@latrobe.edu.au

Ben Soh
Department of Computer Science and Information Technology, School of Engineering and Mathematical Sciences, La Trobe University, Victoria, Australia
b.soh@latrobe.edu.au

*Abstract*-**Ensuring security for lightweight cryptosystems in mobile cloud computing is challenging. Encryption speed and battery consumption must be maintained while securing mobile devices, the server, and the communication channel. This study proposes a lightweight security protocol called FEATHER which implements MICKEY 2.0 to generate keystream in the cloud server and to perform mobile device decryption and encryption. FEATHER can be used to implement secure parameters and lightweight mechanisms for communication among mobile devices and between them and a cloud server. FEATHER is faster than the existing CLOAK protocol and consumes less battery power. FEATHER also allows more mobile devices to communicate at the same time during very short time periods, maintain security for more applications with minimum computation ability. FEATHER meets mobile cloud computing requirements of speed, identity, and confidentiality assurances, compatibility with mobile devices, and effective communication between cloud servers and mobile devices using an unsafe communication channel.**

***Keywords-mobile cloud computing; lightweight encryption; battery consumption; offloading tasks; MICKEY 2.0***

## I. INTRODUCTION

Data transfer between two mobile devices and from a mobile device to the cloud should be done securely, through multiple different communication channels, such as Wi-Fi, 4G and 5G. A secure protocol for data transfer through unsecure communication methods is required. As mobile devices have limited computation power, it can be difficult to address all security cryptosystem tasks. Authors in [1] proposed a mobile cloud computing enterprise that consists of mobile devices, a wireless core, Wi-Fi access points, and regional information centres. In addition to the limited computational capabilities of mobile devices, battery consumption due to heavy computations adds another challenge. Authors in [2] showed that mobile computing could save energy, such as battery life and wireless energy, by offloading some tasks to a cloud server, which is used to transfer the data in some applications, however some applications are not energy efficient.

To meet security challenges, as well as the demand for a lighter security protocol to save time and address computation power, device hardware limitations, and battery consumption, the research questions to answer are:

- How can MICKEY 2.0 be implemented efficiently to secure communication between mobile devices in mobile cloud computing?
- How can the performance of a new security protocol be evaluated against the existing protocols?
- How can the claim that the proposed protocol is immune from attack be justified?

The aims of this research are:

- To implement MICKEY 2.0 efficiently to secure communication between mobile devices in mobile cloud computing.
- To evaluate the performance of the new security protocol against the existing protocols.
- To provide a clear justification that the new security protocol is immune from possible attacks.

This paper proposes a new protocol, FEATHER, to better meet the security and energy needs of mobile cloud computing and mobile devices than existing protocols.

## II. BACKGROUND

### A. *Mobile Computing*

Mobile cloud computing serves important applications, such as mobile learning, mobile commerce, mobile gaming, eHealth applications, and web searching, [3] and is growing at a fast rate, with 4.78 billion mobile devices globally predicted by the end of 2020 [4]. With many devices connected to each other via large networks, there is a vulnerability to attacks that requires the use of reliable security protocols. Encryption systems suitable for these devices on insecure communication channels are needed. A secure communication protocol must meet the following requirements: speed, identity protection, confidentiality, compatibility with mobile devices, and effective communication between cloud servers and mobile devices through a communication channel that is not safe. Many cryptosystems meet the demand for private and secure

Corresponding author: Ahmed Alamer

transfer of confidential information. However, some require a large computation capability. The Advanced Encryption System (AES) is widely used because it is a very strong and secure cryptosystem [5]. However, it is a "heavy" system that requires large computational resources, has high power consumption, and is therefore not suitable for mobile devices with limited computation capacity. Some researchers have introduced lightweight versions of AES for small devices, such as ALE [6], to reduce the demand on resources, such as Central Processing Units (CPUs) and memory, used to generate the keystream. Some components in cloud computing such as embedded systems on cloud computing with 32-bit, 16-bit, and 8-bit microcontrollers cannot meet real time demands for conventional methods of cryptography [7]. Therefore, AES is a poor solution for many embedded devices in cloud computing that have low computation ability.

### *B. Cloud Computing*

For a lighter encryption method in cloud computing, lightweight stream ciphers can be implemented to provide the required security. Lightweight stream ciphers include a decryption function and an encryption function to handle messages of arbitrary length. Thus, they are better than block ciphers, such as AES, that only handle inputs of a fixed length. Due to their functionalities, they are well adapted to low bandwidth or noisy communications and thus are appropriate for cloud computing. Speed, memory, number of CPUs, and cost efficiency are also important factors [8]. In [9], a MICKEY 2.0 variant, MICKEY 2.0.85, was proposed as the preferred choice over other lightweight stream ciphers. It is lighter and has lower energy consumption, which means it is more cost efficient [10]. However, the protocol can be adapted to implement other lightweight ciphers, such as Trivium or Grain. Al-Omari [11] proposed a lightweight block cipher-based encryption mechanism and tested a faster algorithm by comparing it to an AES cipher in terms of speed. Ali et al. [12] focused on a cloud-based file distribution and management model, and showed that the ability of cloud computing to adapt is important for users, and not only in terms of data storage. The study also addressed the problem of offloading tasks to the server by using multiple servers and demonstrated how this method provides more security when sharing data. Hassan et al. [13] discussed cloud computing applications using machine learning approaches as a useful direction for predicting loading using statistical analysis, as well as for ensuring service level agreements.

## III. LITERATURE REVIEW

Bahrami and Singhal [14] studied the adequacy of using AES in mobile cloud computing and explained that, due to cost, cryptosystems such as AES are not suitable for mobile devices, because mobile devices have limited resources, such as limited power energy, low speed processors, and tiny RAM capacity. AES is not the appropriate encryption technique, since offload and download is done for every single transferred file. They introduced lightweight methods, such as pseudo-random permutation, based on chaos systems. Another solution is using lightweight security methods that provide a balance between energy efficiency and security. A lightweight security technique can be considered an easy operation (i.e. a permutation) instead of using complicated and expensive operations when using secret key or public key encryptions [15-17].

### *A. The Advantage of using Stream Ciphers in Small Devices*

A stream cipher is a symmetric cryptosystem that uses the same key for encryption and decryption. Stream ciphers can transform data faster than other ciphers, such as block ciphers, and also faster than ciphers in an asymmetric cryptosystem [18, 19]. Stream ciphers are less secure than other, symmetric and block cipher, types of cryptosystems, such as AES which is one of the most secure ciphers. The encryption process in AES involves permutations and a substitution process and requires a number of rounds, which increases the power and storage requirements. On the other hand, lightweight stream ciphers such as MICKEY 2.0, Trivium, and Grain [20] need much less power and memory. Widely used lightweight stream ciphers for small applications include E0 (Bluetooth), RC4 (Web), and the A5 family (GMS) [21]. Stream ciphers have advantages due to their high throughput property and low computational complexity. Lightweight stream ciphers [22] are a better choice than block ciphers because they need less memory and less hardware complexity

### *B. Using Lightweight Stream Ciphers in Cloud Computing and Mobile Cloud Computing*

Lightweight stream ciphers have several advantages for cloud computing. They provide fast encryption by generating a secure keystream faster than other popular ciphers, such as AES. They need fewer computation facilities, such as CPUs and memory from the cloud, which reduces cost and power consumption significantly. Additionally, they include faster encryption, the consumption of less battery power, and lower bandwidth requirements.

### *C. AES and CLOAK Protocols*

CLOAK is a lightweight protocol based on the AES cipher that enables two mobile devices to communicate, while leaving the keystream generation on an external server [23]. As CLOAK can get the keystream from either trusted or untrusted external servers, the main security concern is to protect the keystream. Security can be compromised by fetching the keystream from an external server and from communication media. Lightweight stream ciphers that can be used in mobiles include Trivium [24], Grain [25], and MICKEY 2.0 [26]. MICKEY 2.0 cipher is more resistant to statistical attacks [27-29] and it can produce large throughput. The lightweight protocol developed in this study does not rely on the server to be secure and will not be compromised as in the CLOAK protocol, which assumes the security of the server relies on the server provider [23]. Using MICKEY 2.0 in this lightweight protocol to provide a secure keystream is significantly faster than using AES. For example, the time needed by the server to generate the keystream is reduced, which in turn reduces the time to transfer the data between the server and the mobile.

Adithya et al. [30] introduced another enhancement for the CLOAK security, which is compared to the proposed FEATHER protocol in this study in Figure 3, and discussed in Section VIII.

## IV. THE LIGHTWEIGHT PROTOCOL FEATHER

### *A. Overview*

The study designed a MICKEY 2.0 cipher based protocol called FEATHER to strengthen confidentiality and protection during messaging between mobile devices, as well as communication between devices and the cloud server (see Figure 1). The MICKEY 2.0 cipher produced a secure keystream in the external server to reduce reliance on mobile devices that have limited computing power and memory. The role of mobile devices is only encryption and decryption, which allows mobile devices to compute and reduce the amount of energy consumed by the device battery.

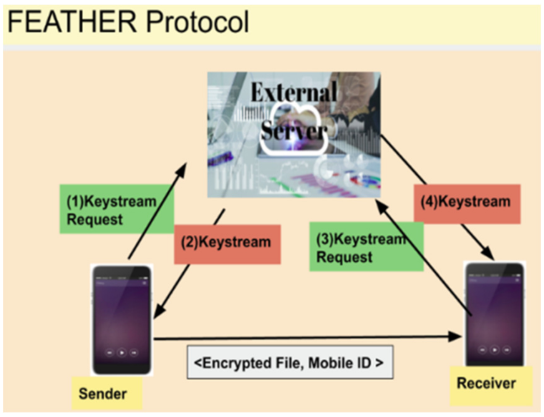

Fig. 1. Mobile cloud basic communication.

A lightweight secure protocol is introduced for communication between devices and the external server over the cloud, as well as design applications on mobile devices for the process of verification and encryption and decryption. The proposed protocol is faster and can move larger files than the CLOAK cipher [23]. The protocol also maintains a high level of security. The protocol was designed to achieve security through the application of the MICKEY 2.0 cipher with additional protection systems for identity verification, such as hash functions, time stamps, and out-of-band passwords. A lightweight stream cipher is needed to generate the keystream faster and use fewer resources, so more secure applications can take advantage of advances in mobile cloud computing. If the keystream generated in the server is faster, it will allow more mobiles to get it from the cloud compared to a heavy encryption system like AES. Thus, it will be more efficient and will reduce cost. Using MICKEY 2.0 meets most of these needs.

### *B. Design Principles*

There are ten design principles for a lightweight protocol:

#### *1) Avoid Implementing a Heavy Encryption Method*

As some popular encryption algorithms, such as AES, require considerable resources in terms of CPU time and/or memory usage, the protocol should offload the more computing-intensive steps to a server in the cloud while simplifying the steps carried out on the mobile device. Therefore, a lightweight protocol can offload generation and storage of the keystream to a server using the MICKEY 2.0 algorithm.

#### *2) Avoid Relying Entirely on the Server*

It is important to avoid relying entirely on the server to ensure communication security. Then, even if an adversary compromises the server, it cannot easily use the captured keystream data to decrypt messages directly. Although the client receives a keystream from the server, the client does not use it directly. Instead, the client selects a few random values using primitive polynomials to apply the keystream to the plaintext to compute the encrypted data.

#### *3) Send Messages between Client and Server over the Internet*

The protocol must assume an adversary may intercept messages or an impostor may try to insert invalid messages in the client–server communication. One popular approach is to use a key-exchange algorithm, such as Diffie–Hellman (which is vulnerable to a man-in-the-middle attack), or a more sophisticated Station-to-Station protocol [31], which avoids this vulnerability. The significant computation of these approaches may not be appropriate for simple mobile or microcontroller devices. The protocol needs to assume the ability to send brief out-of-band messages using a different communication medium. For example, if the protocol is implemented on top of the HTTP protocol, a secret out-of-band message may be sent by email or SMS. In this protocol, an out-of-band message is sent from the server to the client to convey a one-time-pad, and from one client to another to convey a file token and the secret values (using primitive polynomials) used to step through the keystream.

#### *4) Focus on Authentication on Unique Security Parameters*

For authentication, this protocol uses a “bring something, know something” technique. The protocol assumes that each mobile device (or microcontroller device) has a Universally Unique Identifier (UUID). It also allows each user to select a username that is not necessarily unique. These are combined using a hash function to generate a Unique Identifier (UID) for each user. At the initiation of the protocol, each user registers its UID and then communicates an encrypted copy of its secret password to the server. For subsequent communication, all messages between the client and the server are validated using a digital signature based on hashing the message and the secret password. In this case, the “bring something” refers to the device and its UUID and the “know something” refers to the user’s secret password. Since an adversary does not know the secret password, it cannot generate a valid signature, so the client and server can reject messages with invalid signatures.

#### *5) Secure the Communication between the Client and the Cloud Server*

Client-server communication security relies on a shared keystream. This shared keystream is first generated by the server when the client sends a message to register the user. In its response to the client, the server sends the shared keystream, encrypted with the one-time-pad, to prevent an adversary from capturing the keystream.

### *6) Offload the Keystream Generation to the Cloud Server*

The server implementation may use any reasonable technique for keystream generation. In practice, a method is needed that is computationally efficient and still provides a reasonable level of security. To generate a new keystream for each user, the server must first create an initial key (or key+IV pair).

### *7) Ensure Client Request for the Keystream from the Cloud Authentications*

When the client submits a request to generate a new keystream, it includes a token and expiry time. There are two possible implementations. The server may simply generate and store a key, and then generate the actual keystream "on the fly" whenever requested. Alternatively, the server may generate the keystream right away and store it as a file to be retrieved later when the client submits the corresponding token. The expiry time allows the client to limit the time the keystream is stored on the server. This reduces the availability of the keystream if an adversary tries to compromise the server.

### *8) Ensure There Are Possible and Flexible Variations for Secure Data Transfer*

To enhance the security of the protocol, the server never has access to the unencrypted data. The data are encrypted by the client, using a modified version of the keystream, and this modification is unknown to the server. When transferring encrypted data from one client to another, there are three main options available.

- In one variation, since the data are securely encrypted, the file can be uploaded to any simple file server. This may provide an increased level of security since it introduces a separation from the keystream server and the file server. In fact, clients would be free to use a variety of different file servers to transfer encrypted data files, as long as these are communicated between the sender and receiver.
- In a simpler implementation, the clients can upload or download the encrypted data to the server and are being identified by a unique token which can be pseudo-randomly generated. Any other client can download the encrypted file, asynchronously, once it receives the appropriate token from the first client. Some efficiency can be gained if the file upload and download is implemented on the keystream server, since the same protocol mechanism can be used to download a keystream (given a token) or to download encrypted data (given a token). In fact, once a keystream is generated and stored as a file, the keys used to generate the keystream could be deleted, reducing the vulnerability of the protocol.
- In the third option, the encrypted data could also be transferred directly and synchronously from one client to another. This approach could make sense when a pair of clients wants to send and receive a number of smaller messages, as in a secure chat session. This can be accomplished first by generating and downloading a keystream and then sending encrypted messages back and forth without requiring an intermediate file server.

### *9) Modify the Keystream to Further Enhance Security*

For efficiency, the client uses a keystream generated by the remote server, but for security, the keystream is modified in a way unknown to the server. In particular, the client randomly selects a few parameters that describe a particular pseudo-random permutation of keystream values. By sharing these secret permutation parameters with the other client through an out-of-band communication, the other client will be able to decrypt the encrypted file.

### *10) Ensure Data in the Cloud Server are Tied to Expiry Time*

The security of the protocol is enhanced by reducing how long information is retained before being deleted. The keystream and the encrypted files have an associated expiry time, after which the server deletes them. This reduces the information that is exposed if the server is compromised.

## *C. Algorithmic Demonstration of the FEATHER Protocol*

### *1) Channels*

1. Insecure channel e.g Internet HTTP
2. Out-of-band channel e.g SMS

### *2) Algorithm 1: Mobile Device*

```
Step 1: Register mobile with server
i. Pick a unique username
ii. Create UID: Hash (username, device ID)
iv. Get the timestamp t
v. Send register action (via channel 1) with payload [mobile phone number, UID, t]
vi. Wait for response
vii. If OK status received, go to step 2. Otherwise, ERROR status received, go to step 1 (ii).

Step 2: Update password with server
i. Wait for one-time-pad, OTP
ii. Provide a password
iii. Create hashed password, pass: Hash (password, UID)
iii. Create an encryption d: XOR (pass, OTP)
iv. Get the timestamp t
v. Create the payload, x: Hash (UID, d, t)
vi. Send update action with payload
vii. Wait for response
viii. If OK status received, go to step 3. Otherwise, ERROR status received, go to step 2 (ii).

Step 3: Validate password with server, if the first time
i. Send validate action with payload x
ii. Go to step 4.

Step 4: Generate keystream from server
i. Provide a unique 32-byte token
ii. Send generate action with no payload
iii. Wait for keystream response, with bytes size n
iv. Go to step 5.

Step 5: Share keystream with another mobile
i. Specify expiry time, e
ii. Get the timestamp t
```

```
iii. Create a unique token: Hash (UID, e, t)
iv. Create an encryption f: XOR (token, keystream)
v. Create the payload, x: Hash (message, pass, UID, e, t, f, n)
vi. Send payload x (via channel 2)
vii. Go to step 6.

Step 6: Upload to server
i. Provide a 32-byte file-id
ii. Create a file: Hash (UID, file-id, e, t)
iii. Create an encryption f: XOR (file, keystream)
iv. Create an encryption d: XOR (file-contents, token, keystream)
v. Send upload action with payload [UID, f, d].

Step 7: Request from server
i. If token + keystream, create an encryption f: XOR (token, keystream). Otherwise: an encryption f: XOR (file-id, keystream)
ii. Get the timestamp t
iii. Create the payload x: Hash (pass, UID, f, t)
iv. Send request action with payload x.
```

*3) Algorithm 2: Server*

```
Step 1: Wait for registration request from mobile
i. Receive registration action with UID:
ii. Get the timestamp t
iii. If no account with the UID exists:
    a. Create a new account
    b. Respond with [OK status, t]
    c. Send one-time-pad (via channel 2)
    d. Go to step 2
    Otherwise, account with UID exists:
    a. Respond with [ERROR status, error code, t]
    b. Go to step 1.

Step 2: Wait for update request from mobile
i. Receive update action with encrypted payload, d, and signature
ii. Recompute the signature
iii. Decrypt the hashed password, pass
iv. Validate the message
v. If message is valid:
    a. Respond with [OK status]
    b. Go to step 3. Otherwise:
    a. Respond with [ERROR status]
    b. Go to step 2

Step 3: If process validate request received (only first time)
i. Go to step 2 (ii). Otherwise, go to step 4.

Step 4: Wait for generate request from mobile
i. If no payload, generate keystream:
    a. Generate random MICKEY 2.0 keystream
    b. Respond with the key
    c. Go to step 4. Otherwise, payload received:
    a. Create hashed payload: Hash (payload, keystream)
    b. Store hashed payload
    c. Go to step 5.

Step 5: Wait for upload request from mobile
i. Receive upload action with encrypted file payload
ii. Store the file
iii. Respond with [OK status]. Otherwise something goes wrong, respond with [ERROR status].

Step 6: Wait for 'request' request from mobile:
i. Receive request action with encrypted token or file-id
ii. Lookup the token and create the requested data d: XOR (token + keystream). Otherwise d: XOR (file-contents, keystream)
iii. Get the timestamp t
iv. Create the payload x: Hash (pass, d, t, OK status)
v. Respond with payload x
```

## V. Protocol Implementation

The FEATHER communication protocol enables mobile devices with limited computational resources to share encrypted files with the help of an external server that has greater computing, storage, and bandwidth resources. The protocol uses two communication channels. The first channel is assumed to be insecure, such as the Internet using HTTP to transport messages between the mobile devices and the external server. The second channel carrying "out-of-band" messages is assumed to be secure and could be implemented using SMS messages to mobile devices, or possibly email. The first channel allows mobile devices to initiate six actions by sending a message to the external server and receiving a response. The second out-of-band channel is used to send and receive three kinds of secret information:

- A one-time-pad, which could use a more secure parameter instead of justification.
- A file id.
- A token id (and some additional parameters).

The protocol also uses a cryptographic hash function, such as SHA-256, which outputs a 32-byte hash value. For distinct pairs of strings, s and t, $H(s) \neq H(t)$ (with very high probability). Messages in the protocol are simply concatenated key = value pairs of parameters. Each of the 11 possible parameters is identified by a unique character:

a = action
s = status
c = code (error code)
u = uid
p = phone
f = token or file
d = data
n = number
e = expire
t = timestamp
x = signature

The timestamp is Unix time in seconds, and can help prevent "replay attacks". The cryptographic signature is a hash of the entire message string (before the signature is added) and is used to authenticate messages. The six actions and messages are: REGISTER, UPDATE, VALIDATE, GENERATE, UPLOAD, and REQUEST. Figure 2 illustrates the secure communication between the basic components, server, mobile devices and communication channel.

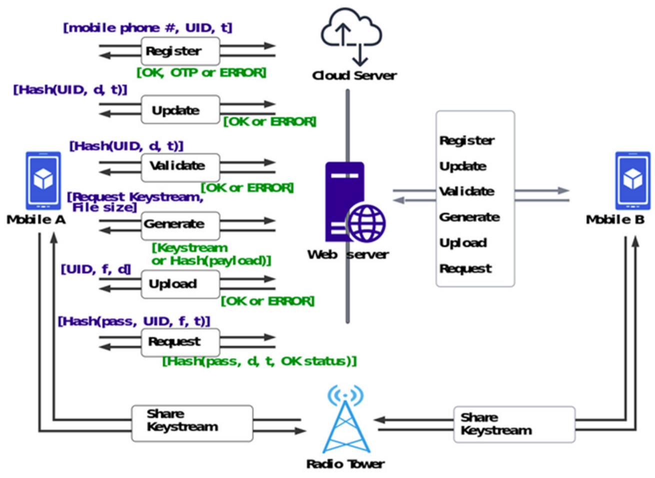

Fig. 2. FEATHER protocol message communications between the mobile devices and the cloud server.

## A. REGISTER

The person using the mobile device app provides a username (e.g. "Jason"). The device hardware is also assumed to have a unique hardware identifier (e.g. Device ID). The mobile app combines these strings using a hash function to get a unique id that can be sent to the external server without revealing any private information.

uid = H(device ID, username); 32-byte value.

The mobile device also has a telephone number at which it can receive an out-of-band message via SMS. The person registers an account on the external server by sending a message:

a = register
u = uid
p = phone
t = timestamp

When the external server receives this message, if no account exists for that uid, a new account is created, and this message is sent back:

s = OK
t = timestamp

If an account already exists for that uid the server responds:

s = ERROR
c = code (indicating type or error)
t = timestamp

If an account already exists for the given uid, the person needs to pick a new username to create a different uid:

ONE-TIME-PAD via SMS

Following a successful REGISTER message, the external server sends a one-time-pad to the mobile device via an out-of-band channel using SMS to the phone number provided. The person would need to cut-and-paste this string into the mobile device app to be stored.

## B. UPDATE

In the mobile device app, the person also provides a password (e.g. "MySecret"), which provides a type of "bring something, know something" security (bring something = mobile device, know something = username, password). The user's simple password is combined with the uid to create a "hashed password", which will be sent to the external server.

pass = H(uid, password)

The hashed password is encrypted using XOR with the secret one-time-pad. The entire message (before the signature) is hashed to create a cryptographic signature for authentication.

a = update
u = uid
d = XOR(pass, OTP(One-Time-Pad))
t = time
x = H(message)

The external server confirms the validity of the message by recomputing the signature, and then decrypts and stores the hashed password in the account. The response is either OK or ERROR.

## C. VALIDATE

This message is optional but useful for debugging purposes when implementing this protocol for the first time. The mobile device sends the following message asking the external server to confirm that the hashed password and signature are valid.

a = validate
u = uid
d = XOR(pass, OTP)
t = time
x = H(message, pass)

The external server decodes the hashed password, recomputes the signature, and responds with either OK or ERROR.

## D. GENERATE

The mobile device provides a unique 32-byte token and asks the external server to generate a new encryption key that will be used to generate a keystream of "number" bytes that will be stored until a given "expire" time. The unique token is created by hashing the uid, expire, and timestamp:

token = H(uid, expire, timestamp)

The token is XOR-encrypted with the shared-keystream. The message sent to the server has these parameters:

a = generate
u = uid
f = XOR(token, shared-keystream)
n = number (of bytes in the keystream)
e = expire
t = timestamp
x = H(message, pass)

The external server generates a random MICKEY 2.0 key (20 bytes of key+IV). There are two implementation-dependent choices:

- The server can simply store the 20-byte in association with the token and generate the keystream on-the-fly when requested, or
- The server can generate and store the keystream and then discard the 20-byte key. With this option, the token becomes equivalent to a file-id, and the keystream becomes equivalent to the file contents.

### E. UPLOAD

The mobile device asks the external server to store a file by providing a 32-byte file-id, the encrypted contents of the file, and an expiration time, after which the file will be deleted. The unique file-id is created by hashing the uid, filename, expire, and timestamp:

file = H(uid, filename, expire, timestamp)

The file-id is XOR-encrypted with the shared-keystream. The mobile device sends a message with these parameters:

a = upload
u = uid
f = XOR(file, shared-keystream)
d = XOR(file-contents, token-keystream)

The external server stores the file and responds with OK or else ERROR if something goes wrong.

### F. REQUEST

A mobile device can request a token-keystream or encrypted file contents by providing the appropriate 32-byte token or file-id. The message has these parameters:

a = request
u = uid
f = XOR(token, shared-keystream)
or f = XOR(file, shared-keystream)
t = timestamp
x = H(message, pass)

The external server uses the token (or file-id) to look up the requested data and sends it back to the mobile device.

s = OK
d = XOR(token-keystream, shared-keystream)
or d = XOR(file-contents, shared-keystream)
t = timestamp
x = H(message, pass)

The protocol assumes the first mobile device (the sender) is able to communicate the “token” and “file” to the second mobile device (the receiver) through a secure out-of-band channel, here assumed to be via an SMS message. It is important that the communication remains secure even if the external server is compromised by an adversary. Therefore, the token-keystream is not used directly to encrypt the file contents, since someone with access to the server could easily decrypt the file. Instead, the first mobile device must pick several random numbers $R_1$, $R_2$, $R_3$, ... that are used to walk through the bytes of the token-keystream in a deterministic but difficult to predict order. These sets of random numbers must also be communicated to the second mobile device through a secure out-of-band channel. For example, for a token-keystream with length $N=2^k-1$, which is a prime number, the index of the next byte to be used could be calculated as:

$\text{index}(i) = R_1 \bmod N$
$\text{index}(i+1) = (R_2 * \text{index}(i) + R_1) \bmod N$

The mobile app was designed in Android Studio and then the app was transferred as a file to be converted into a mobile local app. The code was written in Java on the Android studio platform, which works on all major operating systems (i.e. Windows, MacOS and Linux).

## VI. RESULTS AND ANALYSIS

The performance of FEATHER protocol is measured on two items: the overall speed and battery consumption.

### A. Speed Performance

Five different mobile devices with Android-based operating systems, shown in Table I, were used to test the protocol performance. The total time from downloading the keystream, encryption, and writing to storage was measured. Tables II–V show the computations in five different Android-based devices.

TABLE I. SPECIFICATIONS OF THE USED MOBILE DEVICES

| | D-1 | D-2 | D-3 | D-4 | D-5 |
|---|---|---|---|---|---|
| **Model name** | LG V20 | Huawei Nova 3e | Samsung Galaxy S9+ | Samsung Galaxy A6+ | Lenovo M10 Tablet |
| **OS** | Android 7.0 Nougat | Android 8.1 with EMUI 8.0 | Android 9.0 P | Android 8.0 Oreo | Android 8.0 Oreo |
| **API level** | 24 | 26 | 28 | 26 | 27 |
| **CPU** | Quad-core 2.15GHz + 1.6GHz | Quad-core 2.36GHz | Octa-core (4×2.7GHz & 4×1.7GHz) | Octa-core 1.8Ghz | Octa-core 1.8GHz |
| **Chipset** | Qualcomm Snapdragon 820 | HiSilicon Kirin 659 | Qualcomm Snapdragon 845 | Qualcomm Snapdragon 450 | Qualcomm Snapdragon 450 |
| **RAM** | 4GB | 4GB | 6GB | 4GB | 3GB |
| **GPU** | Adreno 530 | Mali-T830 MP2 | Adreno 630 | Adreno 506 | Adreno 506 |
| **Battery** | 3200mAh, Li-Ion | 3000mAh, Li-Polymer | 3500mAh, Li-Ion | 3500mAh, Li-Ion | 4,850mAh, Li-Ion Polymer |

Table II shows the total time average for the five different devices. The LG V20 device was the slowest at 18.44169s. However, it was very fast for 8MB file size. The Samsung Galaxy S9+ device had the fastest total time average (for Download, Decode and Write) at 10.3438s. The total time for all five devices was 71.6456s and the average was 14.3291s. In the experiments, 15 different file sizes from 1KB to 16MB were used to measure the overall performance, as shown in Tables III-V. It is clear that FEATHER can handle large files, and 16MB file size is sufficient to transfer documents and photos. These calculations use the Samsung Galaxy S9+, and a 16MB file only needs about 19.0s overall time which includes downloading the encrypted file from the external server, decryption time and storing it to the device (write).

TABLE II. RUNNING 8MB FILE 60 TIMES AND AVERAGE TIME (S)

| | D-1 | D-2 | D-3 | D-4 | D-5 |
|---|---|---|---|---|---|
| Down load | 18.0833 | 11.594383 | 10.162450 | 17.28501 | 13.083 |
| Decode | 0.13299 | 0.090583 | 0.0872030 | 0.151201 | 0.1143 |
| Write | 0.22536 | 0.12371666 | 0.0941666 | 0.2327666 | 0.1841 |
| Total time | 18.44169 | 11.80868266 | 10.3438196 | 17.668978 | 13.382 |

TABLE III. RUNNING 3 TO 512KB FILES AND CALCULATING TIME (S)

| File size | 32KB | 64KB | 128KB | 256KB | 512KB |
|---|---|---|---|---|---|
| Download | 0.324 | 0.38 | 0.424 | 0.743 | 1.001 |
| Decode | 0.00102 | 0.0015886 | 0.0023127 | 0.0085227 | 0.01105 |
| Write | 0.085 | 0.066 | 0.068 | 0.057 | 0.052 |
| Total time | 0.41002 | 0.4475886 | 0.4943127 | 0.8085227 | 1.0640 |

TABLE IV. RUNNING 1 TO 16MB FILES AND CALCULATING TIME (S)

| File size | 1MB | 2MB | 4MB | 8MB | 16MB |
|---|---|---|---|---|---|
| Download | 1.684 | 2.957 | 5.439 | 9.625 | 18.664 |
| Decode | 0.036893 | 0.02132 | 0.0324316 | 0.0836791 | 0.1598017 |
| Write | 0.057 | 0.085 | 0.092 | 0.106 | 0.19 |
| Total time | 1.777893 | 3.06332 | 5.5634316 | 9.8146791 | 19.013801 |

## B. *Power Consumption*

An Android-based application, GSam Battery Monitor [32] was used to measure the overall battery power consumption of FEATHER using a Samsung Galaxy S9+ with a 3500mAh Li-ion battery. After running GSam and the mobile app for FEATHER, the results showed that performing the operations on 10 files varying in size from 2 to 16MB consumed less than 1% of all apps running in the background, which consumed 1% of battery power, so FEATHER consumes only 0.0001% of battery power.

## C. *. FEATHER vs. CLOAK*

The proposed FEATHER protocol is lighter than CLOAK and is much faster. Comparing the performance for file sizes of 1, 2, 4, and 8MB shows that FEATHER is faster. For example, in Table VI, the total time for 8MB file size is 110s for CLOAK and about 9.8s for FEATHER. Therefore, FEATHER is even more practical if multiple devices need to communicate at the same time. In addition, FEATHER consumes 80% less battery power than CLOAK. Adithya et al. [30] presented another secure application of CLOAK protocol in Apache server, using a Graphical User Interface to provide more security, however it takes 1 to 2s for users to enter the digits. FEATHER, CLOAK and [30] are compared in Figure 3.

TABLE V. CLOAK AND FEATHER PROTOCOLS: TOTAL SPEED TIME FOR DIFFERENT FILE SIZES

| File size (MB) | Total time (s) | |
|---|---|---|
| | CLOAK | FEATHER |
| 1 | 20 | 1.77789342 |
| 2 | 30 | 3.06332185 |
| 4 | 60 | 5.56343165 |
| 8 | 110 | 9.81467915 |

## VII. ATTACK ANALYSIS

This section provides an analysis of common attacks and shows how FEATHER is resistant to these types of attacks.

## A. *Man in the Middle Attacks*

The attacker can interrupt data, inject information, and redirect the traffic. This can be between the two devices or between the devices and the external server, thus it works on the communication channel. This can be prevented by providing strong mutual authentication and end point authentication, as the FEATHER protocol does, and by using hashing for messages, so as all messages are wrapped in hash functions. Thus, FEATHER is immune from man in the middle attacks.

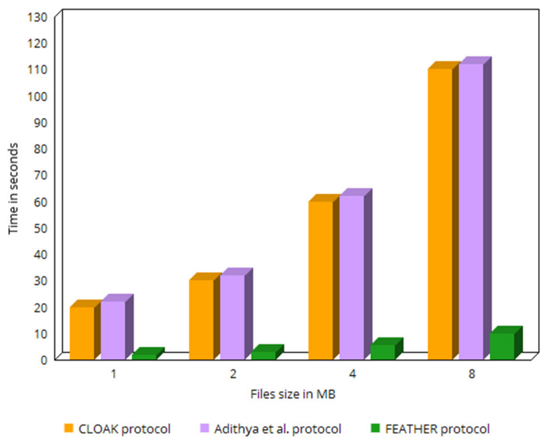

Fig. 3. CLOAK, FEATHER and Adithya et al. [30] speed performance comparison.

## B. *Insider Attacks*

On the server side, if an insider can gain access to the information they only gain access to the keystream. However, the message will be included in a hash function, as well as the one-time-pad, and another secure parameter such as the timestamp or a random number only known by the mobile device users. On the mobile side, the mobile will validate the messages received from the server and other mobiles.

## C. *Denial of Service Attacks*

The FEATHER protocol has steps in the external server to authenticate users before accessing the service by: 1) authentication of users' credentials, 2) updating the accessing parameters, and 3) validating the users' messages and hash functions. As verification by the server and devices is mutual, a denial of service attack is not applicable.

## D. *Chosen IV-Attacks*

The keystream is generated by using MICKEY 2.0 and (key, IV) as the initial input. In FEATHER, the IV is not used more than once with the same key, thus FEATHER eliminates this threat by preventing the reuse of the IV, as well as by including the IV in the hash function, so an attacker choosing the IV will not result in the key being revealed.

## E. *Two-time Pad Attacks*

Assuming there are two messages $m_1$ and $m_2$, if the same key (k) is used (called two-time pad), and there are two ciphertexts ($c_1$, $c_2$) then:

$m_1 \oplus$ k that results in $c_1$ and

$m_2 \oplus$ k that results in $c_2$.

Therefore, it is easy for the attacker to perform the XOR operation for ciphertexts in order to reveal the plaintext as:

$$c_1 \oplus c_2$$

that is, using statistical frequency analysis leads to $m_1 \oplus m_2$. In the FEATHER protocol, each file is encrypted by a different keystream as well as a different one-time-pad for every session and time timestamp. Thus, this attack is not applicable.

### *F. Impersonation Attacks*

This kind of attack occurs when the attacker gains access to a mobile device and requests a response from the server. The server will validate and authenticate the request. As mobile users will be using a hash function, including a one-time-pad (as discussed in the protocol implementation), the server also will hash the keystream with a one-time-pad among other user credentials, meaning this attack is not feasible with FEATHER.

### *G. Brute Force attacks*

As the complexity of a brute force attack in key=80bit in general is $2^{80}$, the FEATHER protocol used a hash function. For example, using D-3 (a user may choose other stronger hash functions, and that will not affect the speed performance as the slower part is the downloading time), the computation power relies on the implementation, and adding other secure parameters such as using OTP, that is similar to the one-time-pad cipher, which substantially raises the computation power needed to break the protocol.

## VIII. Discussion

In FEATHER, downloading is the most time-consuming task compared to the CLOAK protocol. If it is required for more than two mobile devices to communicate at the same time, the external server generating the keystream in the FEATHER protocol is much faster than CLOAK. This will reduce the overall time as the decoding time is just performing XOR on messages with the keystream, which is fast. The mobile battery lifetime is also longer. The proposed lightweight security protocol FEATHER provides confidentiality, authorisation, and security for users in mobile cloud computing technology and IoT technology. It also helps reducing power consumption, which will improve the overall performance of mobile applications. The proposed protocol was analyzed against possible known attacks, which showed that it is secure for implementation. The MICKEY 2.0 cipher was used as a pseudo-random number generator. However, the FEATHER protocol can be adapted to use other IV-based lightweight synchronous stream ciphers. The proposed MICKEY 2.0.85 [9] which is 23% faster in generating pseudo-random numbers, can also be used, however, even using MICKEY 2.0 in FEATHER is fast enough. MICKEY 2.0.85 is useful for other smaller applications. The FEATHER protocol offers a secure contribution to mobile cloud computation. The comparison in Figure 3 shows that FEATHER is much faster than the recent CLOAK and [30] protocols, and it also provides more security.

The limitations of this study include the testing on five devices, although the CLOAK protocol [23] was also tested on five devices. Future work could involve further testing of the performance of FEATHER on a wider range of devices, and compare it to a wider range of existing protocols. Another important direction for future research is adapting other lightweight ciphers such as Trivium, Grain, and other lightweight block ciphers to generate the keystream in the server, and then implementing FEATHER and calculating the overall execution time.

## IX. Conclusion

Ensuring security in mobile cloud computing is critical but challenging. The proposed lightweight security protocol, FEATHER, will reduce cost and time used in the external server. Therefore, it can increase the number of devices communicating at the same time and enhance mobile cloud computing applications. The FEATHER protocol has better performance than existing protocols and can help meet the requirements for secure mobile cloud computing with Internet connectivity.

## Acknowledgment

The authors would like to thank Dr David Jones for his advices on the mobile cloud computing settings.

## References

[1] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, Low Wood Bay, Lake District, UK, Jun. 2012, pp. 21–28, doi: 10.1145/2307849.2307856.

[2] K. Kumar and Y. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer*, vol. 43, no. 4, pp. 51–56, Apr. 2010, doi: 10.1109/MC.2010.98.

[3] H. Dinh Thai, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, Dec. 2013, doi: 10.1002/wcm.1203.

[4] "Number of mobile phone users worldwide 2015-2020," *Statista.* https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/ (accessed Jul. 23, 2020).

[5] G. Singh and S. Kinger, "A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security," *International Journal of Computer Applications*, vol. 67, no. 19, pp. 33–38, Apr. 2013, doi: 10.5120/11507-7224.

[6] A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser, "ALE: AES-Based Lightweight Authenticated Encryption," in *Fast Software Encryption*, Berlin: Springer, 2014, pp. 447–466.

[7] G. Desolda, C. Ardito, H.-C. Jetter, and R. Lanzilotti, "Exploring spatially-aware cross-device interaction techniques for mobile collaborative sensemaking," *International Journal of Human-Computer Studies*, vol. 122, pp. 1–20, Aug. 2018, doi: 10.1016/j.ijhcs.2018.08.006.

[8] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A Survey of Lightweight-Cryptography Implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, Dec. 2007, doi: 10.1109/MDT.2007.178.

[9] Alamer, Soh, and Brumbaugh, "MICKEY 2.0.85: A Secure and Lighter MICKEY 2.0 Cipher Variant with Improved Power Consumption for Smaller Devices in the IoT," *Symmetry*, vol. 12, no. 1, Dec. 2019, doi: 10.3390/sym12010032, Art no. 32.

[10] P. Kitsos, N. Sklavos, G. Provelengios, and A. N. Skodras, "FPGA-based performance analysis of stream ciphers ZUC, Snow3g, Grain V1,

Mickey V2, Trivium and E0," *Microprocessors and Microsystems*, vol. 37, no. 2, pp. 235–245, Mar. 2013, doi: 10.1016/j.micpro.2012.09.007.

[11] A. H. Al-Omari, "Lightweight Dynamic Crypto Algorithm for Next Internet Generation," *Engineering, Technology & Applied Science Research*, vol. 9, no. 3, pp. 4203–4208, Jun. 2019.

[12] M. Ali, N. Q. Soomro, H. Ali, A. Awan, and M. Kirmani, "Distributed File Sharing and Retrieval Model for Cloud Virtual Environment," *Engineering, Technology & Applied Science Research*, vol. 9, no. 2, pp. 4062–4065, Apr. 2019.

[13] M. K. Hassan, A. Babiker, M. Baker, and M. Hamad, "SLA Management For Virtual Machine Live Migration Using Machine Learning with Modified Kernel and Statistical Approach," *Engineering, Technology & Applied Science Research*, vol. 8, no. 1, pp. 2459–2463, Feb. 2018.

[14] M. Bahrami and M. Singhal, "A Light-Weight Permutation Based Method for Data Privacy in Mobile Cloud Computing," in *3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, San Francisco, CA, USA, Apr. 2015, pp. 189–198, doi: 10.1109/MobileCloud.2015.36.

[15] J. Daemen and V. Rijmen, *The Design of RijndaeL: AES - The Advanced Encryption Standard*. Berlin: Springer, 2002.

[16] D. A. Osvik, J. Bos, D. Stefan, and D. Canright, "Fast software AES encryption," presented at the 17th International Workshop on Fast Software Encryption, Seoul, Korea, Feb. 2010, vol. 6147, pp. 75–93, doi: 10.1007/978-3-642-13858-4_5.

[17] M. Yoshikawa and H. Goto, "Security verification simulator for fault analysis attacks," *International Journal of Soft Computing and Software Engineering*, vol. 3, no. 3, pp. 467–473, 2013, doi: 10.7321/jscse.v3.n3.71.

[18] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Berlin Heidelberg: Springer-Verlag, 2002.

[19] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, "Fast Software AES Encryption," in *Fast Software Encryption*, S. Hong and T. Iwata, Eds. Berlin, Heidelberg: Springer, 2010, pp. 75–93.

[20] M. Robshaw and O. Billet, Eds., *New Stream Cipher Designs*. Berlin, Heidelberg: Springer, 2008.

[21] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. Boca Raton, Florida: CRC press, 2018.

[22] L. Diedrich, P. Jattke, L. Murati, M. Senker, and A. Wiesmaier, "Comparison of Lightweight Stream Ciphers: MICKEY 2.0, WG-8, Grain and Trivium," 2016.

[23] A. Banerjee, M. Hasan, M. A. Rahman, and R. Chapagain, "CLOAK: A Stream Cipher Based Encryption Protocol for Mobile Cloud Computing," *IEEE Access*, vol. 5, pp. 17678–17691, 2017, doi: 10.1109/ACCESS.2017.2744670.

[24] C. De Canniere, "Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles," in *Information Security*, vol. 4176, Berlin, Heidelberg: Springer, 2006, pp. 171–186.

[25] M. Hell, T. Johansson, and W. Meier, "Grain: A stream cipher for constrained environments," *IJWMC*, vol. 2, no. 1, pp. 86–93, Jan. 2007, doi: 10.1504/IJWMC.2007.013798.

[26] S. Babbage and M. Dodd, "The MICKEY Stream Ciphers," in *New Stream Cipher Designs: The eSTREAM Finalists*, M. Robshaw and O. Billet, Eds. Berlin, Heidelberg: Springer, 2008, pp. 191–209.

[27] M. S. Turan and A. Dog, "Detailed Statistical Analysis of Synchronous Stream Ciphers," presented at the SASC 2006 - Stream Ciphers Revisited, Leuven, Belgium, Feb. 2006.

[28] S. Al Hinai, L. M. Batten, and B. Colbert, "Mutually Clock-Controlled Feedback Shift Registers Provide Resistance to Algebraic Attacks," in *Information Security and Cryptology*, vol. 4990, Berlin, Heidelberg: Springer, 2008, pp. 201–215.

[29] A. R. Kazmi, M. Afzal, M. F. Amjad, H. Abbas, and X. Yang, "Algebraic Side Channel Attack on Trivium and Grain Ciphers," *IEEE Access*, vol. 5, pp. 23958–23968, 2017, doi: 10.1109/ACCESS.2017.2766234.

[30] V. Adithya, R. Ramya, D. V. Kumar, and M. M. Krishnan, "Cloak Encryption in Apache," *International Journal of Advance Research and Development*, vol. 3, no. 3, pp. 184–187, 2018.

[31] S. Anand and V. Perumal, "EECDH to prevent MITM attack in cloud computing," *Digital Communications and Networks*, vol. 5, no. 4, pp. 276–287, Nov. 2019, doi: 10.1016/j.dcan.2019.10.007.

[32] "GSam Battery Monitor – Apps on Google Play." https://play.google.com/store/apps/details?id=com.gsamlabs.bbm&hl=en_AU (accessed Jul. 23, 2020).