Pervasive and Mobile Computing 36 (2017) 57–67

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

Fast track article

Anonymous end-to-end communications in adversarial mobile clouds

Claudio A. Ardagna [a], Kanishka Ariyapala [b], Mauro Conti [c,*], Cristina M. Pinotti [d], Julinda Stefa [e]

[a] University of Milan, Italy
[b] University of Florence, Italy
[c] University of Padua, Italy
[d] University of Perugia, Italy
[e] Sapienza University of Rome, Italy

ARTICLE INFO

Article history:
Available online 28 September 2016

Keywords:
Anonymity
Mobile cloud computing
Wireless network

ABSTRACT

Today's mobile devices have changed the way we interact with technology. Internet, cloud access, online banking, instant messaging, and file exchange through the cloud are just a handful of the myriad of smartphone services that we make use of every day. At the same time, the very enablers of these services – mobile internet providers and cloud platforms that host them – pose several threats to the anonymity of our communications. In this paper, we consider the problem of providing end-to-end anonymous communications and file exchange under the cooperative privacy threat of involved parties including network operators and cloud providers, which actively tamper with the communication. We propose a solution for delay-tolerant applications (similar to Whatsapp or Email) and prove the security properties of the protocol under this strong attack model. Finally, we present an experimental analysis of the efficiency of our protocol in terms of performance overhead.

1. Introduction

Mobile cloud computing is the paradigm that was built with the goal to save a resource very precious to mobile devices—their battery. The idea is simple: Pushing the execution of (parts of) mobile apps to remote servers residing on the cloud in order to avoid the energetic cost coming from the local execution on the device. The paradigm works best with computation-intensive applications with very limited access to device local resources like sensors, data. In fact, the more computation-intensive a given task, the more the device will benefit from executing it remotely. The less a given task needs to access local resources, the smaller is the device–cloud communication overhead to execute it remotely. Through the years, researchers have proposed offloading frameworks that take smart decisions on what to execute remotely [1–3], and solutions that boost the security of our devices [4–6] or enable efficient data/application backup [7]. Also, solutions that create virtual peer-to-peer networks of smartphone software clones in the cloud enable unprecedented and efficient, complex distributed protocols on mobiles [8,9].

* Corresponding author.
E-mail address: conti@math.unipd.it (M. Conti).

http://dx.doi.org/10.1016/j.pmcj.2016.09.001
1574-1192/

The nature of the mobile apps, but most importantly, their typical complexity, makes it very hard, if not impossible, to use privacy-preserving execution mechanisms like homomorphic schemes. In fact, these mechanisms, designed to operate in hostile environments (e.g., the untrusted cloud) over encrypted user data, are not suitable for application scenarios considering remote execution of mobile apps [10]. While offloading to the cloud, a mobile user has then to fully trust the cloud-side of the process. Not only is the cloud aware of which data and jobs the user is running, but it also knows exactly who is communicating to whom and what information is being shared.

In this paper, we advocate that communication and file-exchange privacy among mobile cloud computing users is achievable, even under very powerful attacks. To this aim, we propose a protocol that, while supporting storage and computation offloading, implements anonymous end-to-end communications for mobile devices in adversarial mobile clouds. Specifically, we consider a strong attack model (Section 3) in which smartphones, cloud clones, the network operator, and the cloud provider are all adversarial entities and can collude to de-anonymize a communication. The cloud provider can both monitor the traffic from/to the user's cloud clones, and have access to the memory within the machines hosting them. Under this powerful and multifaceted attack model, all previous solutions for anonymous end-to-end communication in a mobile cloud computing setting, including ours [11], are unable to provide the requested privacy (Section 2). In this paper we challenge the common belief and come up with a solution that provides anonymity and unlinkability to the users (Section 4). We discuss the security properties of the protocol according to this new challenging attack model (Section 5). We finally investigate on possible slow-down effects in the system and show, through experimental evaluations, that the overhead incurred is affordable (Section 6).

2. Related work

The mobile cloud computing paradigm, though initially designed with the offloading of heavy computations in mind [1–3], brings multifaceted benefits in a large number of application scenarios. It can enable more complex security mechanisms for smartphones [6], or help exploit the cloud to optimize incoming data-traffic, minimize the device connections to remote servers, and ensure efficient data backup in the cloud [4,7,12]. It opens the way to complex peer-to-peer services on mobile devices [5,8,9], otherwise impossible to run on our battery-limited smartphones. All these solutions assume full trust on both the cloud and the network operators providing the device–cloud communication channel. Also, encryption can come to hand for the protection of the user data stored in the cloud. Unfortunately, if the data/application code is encrypted with a key known only to the user, the cloud cannot be exploited for offloading anymore. In addition, encryption does not guarantee full user privacy. Both the cloud and the network operator in fact know how often a user is: (i) Offloading computation to her cloud server (a.k.a. clone of the device [1,8,9]); (ii) storing data on her cloud server; (iii) exploiting the clone as a bridge to communicate/send the data previously stored on it to other users [8]. If the first two issues are unavoidable to achieve all the benefits of cloud computation offloading and backup, the user is increasingly concerned about her privacy when communicating with other users through the cloud.

Wired, wireless, and hybrid networked systems, have always brought the need of anonymous communication protocols [13–20]. Most applicable solutions exploit chains of proxy nodes [21,22], accumulating and forwarding source-encrypted messages in batches. Among them, TOR [22] is probably the most popular one. However, TOR is not applicable in the scenario in this paper because devices and clones on the cloud are uniquely coupled. Also, the communication among two devices directly involves the corresponding clones. If the latter are compromised, they will identify the sender (receiver) even if TOR is employed when communicating with the corresponding clone.

With the increasing popularity of social networks, several works put the trust among friends as a means to achieve anonymity of communications [14–20]. However, these solutions either not fit at all for mobile–cloud computing scenarios, or are computationally heavy for battery-limited devices. To the best of our knowledge, our previous work [11] was the first attempt to address the issue of anonymous communications through the mobile cloud. It provided a user-tunable level of anonymity to sender (indistinguishable among α users) and receiver (indistinguishable among β users), the (α, β)-anonymity, as defined Section 3, in presence of colluding adversaries, including both cloud providers and network operators. The protocol worked under the assumption that the cloud clones of friend users could trust each other, and rely on each other to thwart anonymity breaches of communicating users. Differently from [11], in this work we consider a much stronger attack model: The cloud provider is able to look into a hosted clone's memory and read encryption keys stored therein; other clones, even friend ones, are malicious and can collude with both the cloud provider and the network operator to de-anonymize other user's communication. Our solution also supports computation offloading, in addition to storage offloading, balancing it with data confidentiality.

Other works have addressed a variety of issues in research areas similar to the ones considered in this paper. Senftleben et al. [23] propose a decentralized privacy-preserving microblogging infrastructure based on a distributed peer-to-peer network of mobile users. The infrastructure, using device-to-device communications, is robust against censorship and provides high availability. Daubert et al. [24] present a solution to privacy-preserving sharing of smartphone sensor data and user-generated content via Twitter. The proposed solution ensures both confidentiality and anonymity of users and their messages. Finally, authentication, a milestone in sensitive-data handling platforms like the mobile cloud computing, is exhaustively reviewed in the survey in [25].

3. System and attack models

The goal of our proposal is to achieve (α, β)-anonymity, that is, given a sender s and a receiver r, an adversary Adv should not be able to associate s to less than α users, and r to less than β users. In this section, we present the system and attack models at the basis of our proposal.

System Model. Our system involves different entities, namely *mobile devices* and *standalone apps* belonging to *users*, *cloud providers* hosting *clones* of the devices, *telco operators*, and a supporting *proxy* acting as a middleware between devices and clones. Mobile devices communicate through both the cellular network infrastructure and short-range ad-hoc wireless communication links (we will consider Wi-Fi from now on as a representative technology for this layer). Each device d_k of user k is mapped with a clone c_k (i.e., a virtual machine) in the cloud, as well as with a standalone application std_k in the Internet. Having clones in the cloud is an emergent practice for offloading computations and communications, and for backup purposes. Hence, clones, connected through P2P links in the cloud, are likely to be entities already present and not necessarily introduced for the sake of our protocol. Also, we assume that information on friendship relations involving the system users are freely available (e.g., the public friendship information available in Facebook).

Key Distribution and User Registration. In our setting, all entities in the system (device, clone, cloud provider, standalone application, and proxy) have a private/public key pair and can securely verify the authenticity of others public keys. The clone keys are distributed by the hosting cloud provider, while the device and standalone public/private key pairs are locally generated and then certified by the trusted proxy. To enter the system a user needs to first register its device with the proxy, and certify device and standalone app keys. Then, the user registers its device with a cloud provider of her choice and have a clone assigned to her. During the registration phase the device exchanges the public keys with her clone. Finally, each device d_k shares a secret key SK_k with the corresponding standalone application std_k in the Internet, generated locally on the corresponding device and distributed when necessary through appropriate encryption mechanisms.

The standalone application, the user device, and the proxy are not controlled by the cloud provider. So, their private and secret keys are unknown to it.

Attack Model. We assume a strong adversarial model, where all communication channels in our protocol can be the target of an attack. We consider attacks on wireless communications among devices, communications with the telco operator and proxy, and communications between the clones in the cloud. We also assume different types of adversaries that are either malicious (i.e., possibly diverging by the protocol flow) or just honest but curious (i.e., aiming to violate the privacy, but without tampering with the exchanged messages). In particular, we consider malicious devices, malicious clones, and malicious standalone applications, while we assume honest but curious cellular network operator and cloud provider. Adversaries might collude among them and share their knowledge, such as for instance the device position within the cellular network and keys stored within clones.

Adversaries aim to identify sender s and receiver r of the communication or, in other words, to reduce the anonymity to $(1, 1)$-anonymity. We note that the proxy is trusted and does not collude with any of the adversaries, although our solution is resilient to the scenario in which it is compromised by malicious adversaries [11]. In addition, a device or clone can attack or collude with an adversary to compromise the anonymity of a friend device or clone.

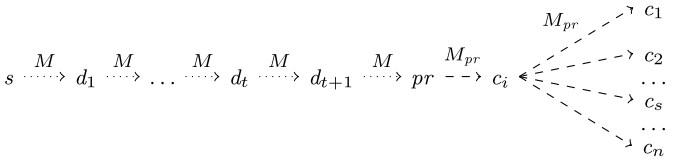
We underline that, when compared to the attack model considered in [11], our work considers a significantly stronger adversary model. In particular, (i) we consider the ability of the cloud provider to look into the memory of the clones and search for encryption keys; (ii) we depart from the assumption of having trusted friend clones (including c_s and c_r); (iii) files can be stored by the clones in the clear.

4. Anonymity protocol

Our solution provides an end-to-end anonymity communication protocol between mobile devices accessing the Internet. We assume a user carrying a mobile device associated with a clone on the cloud, and installing a standalone application supporting anonymity activities on its personal computer. Smartphone data are stored in the clear in the clone and synchronized with it through an encrypted channel. This approach allows to support *full computation offloading* in addition to storage offloading, while reducing as much as possible the parties able to access private data of the users. We note that, although our focus is on (α, β) end-to-end anonymity with support for storage/remote offloading, the offloading can be balanced with data confidentiality as discussed in Section 5. Clearly, the opposite scenario of full computation offloading is that of *full data confidentiality*, which can be provided by encrypting all data stored in the clones at a price of a reduced/nullified computational capability of the clones. An approach *balancing* full computation offloading (all data in the clear to the cloud provider) and full data confidentiality (all data encrypted) can selectively encrypt sensitive data, while storing the remaining data in the clear.

4.1. High-level overview of the protocol

Each communication between sender s and receiver r is composed of three phases as follows [11]: (i) *Sender communication*; (ii) *clone communication*; (iii) *receiver communication*.



$$M=\{[id_{cm}, \alpha, \beta, c_i, c_j, nonce_1, nonce_2]_{K^p_{pr}}, id_f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s}, [c_r, \beta, nonce_2]_{SK_s}\}$$

$$M_{pr}=\{id_f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s}, [c_r, \beta, nonce_2]_{SK_s}\}$$

Fig. 1. Protocol flow for sender communication.

Sender communication implements an anonymous communication between the sender s and corresponding clone c_s, through the proxy and a set of clones. The sender s initially sends its message through a multi-hop WiFi communication on an ad hoc WiFi network of devices. Randomly, a receiving device forwards the message to the proxy using the cellular network. The proxy receiving the message forwards it to a friend clone of c_s, which in turn broadcasts the message to all its friends including c_s. The last communications are carried out on the cloud.

Clone communication implements the part of the communication responsible for anonymously distributing a message between c_s and clone c_r of receiver r. Each friend clone of c_s involved in the sender communication forwards the received message to its standalone app through the Internet. Standalone apps then forward the message to a friend clone, say c_j, of clone c_r via the proxy. Clone c_j finally uses cloud-based communications to broadcast the message to all its friends in the cloud including c_r.

Receiver communication implements the communication between c_r and corresponding receiver r. It is the inverse of the sender communication and involves a proxy, the cellular operator, and a device in the proximity of r. Each friend clone of c_r involved in the clone communication sends the received message to its standalone app through the Internet, which is then forwarded to the proxy. The received messages are filtered by the proxy, which forwards only the real message of s to r via a supporting device (WiFi peer of the destination). The last step uses a mix of cellular and wireless communications.

The following subsections formalize each of the aforementioned high-level phases by presenting, in details, the activities carried out by all the parties involved. Figs. 1–3 summarize the distribution of packets among parties illustrating also the content of each message in all three communication phases. Edges with a dotted line refer to wireless communications carried out on either ad hoc WiFi network (between peers) or cellular network (between peers and the proxy); edges with a dashed line refer to communications over the cloud (between clones and proxy); edges with a solid line refer to communications over the Internet (between clones, standalone apps, and proxy). The edge labels denote the messages exchanged on the corresponding links while the description at the bottom of each figure presents the messages in their entirety.

4.2. Sender communication

Sender communication (Fig. 1) determines the activities carried out in order to anonymously send a message from s to c_s.

User. Similarly to [11], for each communication, user s defines preferences α and β at the basis of the anonymous communication and selects: (i) One friend clone c_i whose social network (S_{c_i}) has at least α members, that is, $|S_{c_i}| \geq \alpha$; (ii) one friend clone c_j of c_r whose social network (S_{c_j}) has at least β members, that is, $|S_{c_j}| \geq \beta$. This selection is done using the friendship database. Then, user s prepares a message M to be sent to c_s that includes: (a) The id id_{cm} of the communication, preferences α and β, the identity of c_i and c_j, and two nonces $nonce_1$ and $nonce_2$ encrypted with K^p_{pr} (the public key of pr); (b) the id id_f of the file to be sent, a number carrying no information neither on the user nor on the device; (c) the identifier id_{cm} of the communication encrypted with K^p_{pr} (the public key of pr); (d) the identity of c_s and $nonce_1$ encrypted with SK_s (the secret key shared between s and its standalone app std_s); (e) the identity of c_r, parameter β, and nonce $nonce_2$ encrypted with SK_s (the secret key shared between s and its standalone app std_s). We note that, to counteract an attack by the cloud provider that aims to uncover the sender s by identifying all clones with less than id_f files, s exploits a concealed file identifier. The same operation is performed by all involved clones to blindly identify the file to be sent according to the protocol. In this way, all selected files will be valid (including the correct file by clone c_s), and the attacker cannot gain any information on the sender. We also note that $nonce_1$ is used to let (i) standalone app std_s know that it is the standalone app of sender s of the communication and (ii) pr distinguish the correct messages among the received ones. Nonce $nonce_2$ has the same role as $nonce_1$ when std_r and r are involved. In addition, it is used to allow replies from r to s over the same anonymized channel (see Section 4.5).

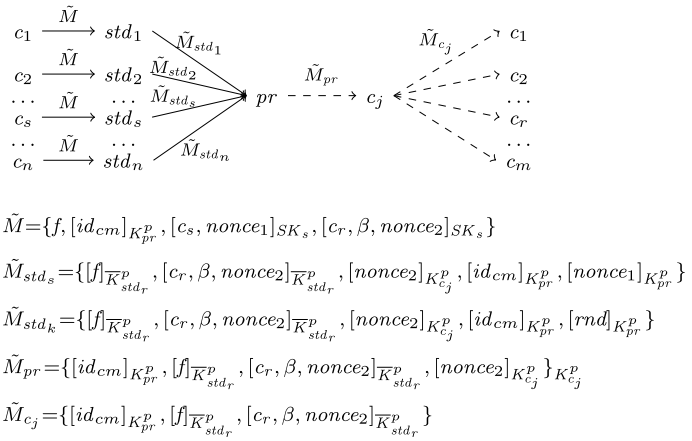


Fig. 2. Protocol flow for clone-to-clone communication.

The message M, prepared by the user as described above and depicted in Fig. 1, is then sent to proxy pr using a probabilistic multi-hop Wi-Fi forward to devices in its physical proximity. To guarantee α anonymity, s sends M *only when* it is surrounded by at least α devices. This process prevents the re-identification from nearby devices [11]. To this aim, all devices periodically broadcast a *probe request* with their identity to surrounding devices.

Device. Upon receiving M, device d_t forwards the message to either another device d_{t+1} in its proximity, or to proxy pr, through the cellular network, using a probabilistic function. The same process is repeated by all involved peers.

Proxy. Upon receiving message M, pr decrypts $[id_{cm}, \alpha, \beta, c_i, c_j, nonce_1, nonce_2]_{K^p_{pr}}$ using its private key K^s_{pr} and stores them for future computations. It then forwards $M_{pr} = \{id_f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s}, [c_r, \beta, nonce_2]_{SK_s}\}$ to c_i.

Clone c_i. It forwards the received message M_{pr} to **all** clones in its social network including c_s. We note that, by sending M_{pr} to all clones, α and β become lower bounds to anonymity, and c_i and c_j behavior is then independent from their value.

4.3. Clone-to-clone communication

Clone-to-Clone communication (Fig. 2) includes all activities aimed to anonymously send a message from c_s to c_r.

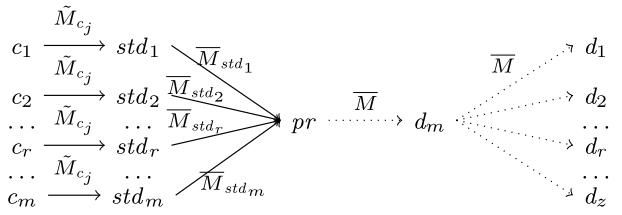
Clone. Each clone c_k receiving M_{pr} blindly identifies the file to be sent by applying a function (e.g., a modulo operation) on the received id_f. It then replaces id_f with f generating a new message $\tilde{M} = \{f, [id_{cm}]_{K^p_{pr}}, [c_s, nonce_1]_{SK_s}, [c_r, \beta, nonce_2]_{SK_s}\}$, and forwards it to the corresponding std_k on the Internet.

Each clone c_k then sends a file in the clear with the same identifier to the corresponding standalone application, showing the same behavior to all observing parties. We note that this approach based on blind file selection is robust to a scenario where the clone c_s is compromised and malicious (see Section 5 for more details). In this case in fact c_s behaves as any other clone in the system and is not able to understand what is going on in the communication, unless it also owns the corresponding standalone app std_s.

Standalone app. Upon receiving $\tilde{M}$, a standalone app first decrypts $[c_s, nonce_1]_{SK_s}$ using its secret key SK_k. If $SK_k = SK_s$, the decrypted ciphertext contains $c_k = c_s$, and std_k identifies itself as std_s, that is, the application of the sender of a communication. std_s decrypts $[c_r, \beta, nonce_2]_{SK_s}$ using its secret key SK_s, encrypts $[c_r, \beta, nonce_2]$ using $\overline{K}^p_{std_r}$ (the public key of std_r), and encrypts $nonce_2$ using $K^p_{c_j}$ (the public key of c_j). It also encrypts f using $\overline{K}^p_{std_r}$ (the public key of std_r) and adds $[id_{cm}]_{K^p_{pr}}$ to the message. Nonce $nonce_1$ is finally added to the new message and encrypted with K^p_{pr} (the public key of pr).

After these activities have been completed, message $\tilde{M}_{std_s} = \{[f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}, [nonce_2]_{K^p_{c_j}}, [id_{cm}]_{K^p_{pr}}, [nonce_1]_{K^p_{pr}}\}$ is generated and sent by std_s to pr. The message sent by $std_k \neq std_s$ involved in the communication is the same as $\tilde{M}_{std_s}$ with the only difference that $[nonce_1]_{K^p_{pr}}$ contains a random number rnd.

Proxy. Upon receiving a message $\tilde{M}_{std_k}$ sent by std_k, proxy pr decrypts the last two fields of $\tilde{M}_{std_k}$ using K^s_{pr}. The first field contains the identifier id_{cm} of the communication to which $\tilde{M}_{std_k}$ belongs, while the second field either $nonce_1$ in case the decrypted message is the correct one ($\tilde{M}_{std_s}$) or a random number rnd otherwise. Upon identifying $\tilde{M}_{std_s}$, the proxy waits until at least α messages belonging to the same communication id id_{cm} are collected. It then prepares message $\tilde{M}_{pr} = \{[id_{cm}]_{K^p_{pr}}, [f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}, [nonce_2]_{K^p_{c_j}}\}_{K^p_{c_j}}$ and forwards it to c_j. We note that waiting for at least α messages



$$\overline{M}_{std_r}=\{[f, nonce_2]_{SK_r}, [id_{cm}]_{K^p_{pr}}, [nonce_2, d_m]_{K^p_{pr}}\}$$

$$\overline{M}_{std_k}=\{[f, nonce_2]_{SK_r}, [id_{cm}]_{K^p_{pr}}, [rnd]_{K^p_{pr}}\}$$

$$\overline{M}=\{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$$

Fig. 3. Protocol flow for receiver communication.

and encrypting the whole message with the public key $K^p_{c_j}$ of c_j forbids re-identification by attackers able to observe the cloud and the standalone apps as discussed in Section 5. We also note that α and c_j are identified using id_{cm} previously stored by the proxy with α, β, c_i, c_j, $nonce_1$, and $nonce_2$.

Clone c_j. Upon receiving message $\tilde{M}_{pr}$, c_j first decrypts it using its private key $K^s_{c_j}$. It then decrypts $nonce_2$ again with its private key $K^s_{c_j}$. We note that $nonce_2$ is used to support bidirectional communications as discussed in Section 4.5. Clone c_j then forwards message $\tilde{M}_{c_j} = \{[id_{cm}]_{K^p_{pr}}, [f]_{\overline{K}^p_{std_r}}, [c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}\}$ to **all** c_k in its social network.

4.4. Receiver communication

Receiver communication (Fig. 3) includes all activities aimed to anonymously send a message from c_r to r.

Clone. Each clone c_k receiving $\tilde{M}_{c_j}$ forwards the message to its corresponding std_k on the Internet.

Standalone app. Upon receiving $\tilde{M}_{c_j}$, a standalone app first decrypts $[c_r, \beta, nonce_2]_{\overline{K}^p_{std_r}}$ using its private key $\overline{K}^s_{std_k}$. If $\overline{K}^s_{std_k} = \overline{K}^s_{std_r}$, the decrypted ciphertext contains $c_k = c_r$, and std_k identifies itself as std_r, that is, the application of the receiver of a communication. std_r decrypts $[f]_{\overline{K}^p_{std_r}}$ using $\overline{K}^s_{std_r}$ (the private key of std_r), and encrypts f and $nonce_2$ using SK_r (the secret key of r). $[id_{cm}]_{K^p_{pr}}$ is then added to the message. Nonce $nonce_2$ and d_m are finally added to the new message and encrypted with K^p_{pr} (the public key of pr).

After these activities have been completed, message $\overline{M}_{std_r} = \{[f, nonce_2]_{SK_r}, [id_{cm}]_{K^p_{pr}}, [nonce_2, d_m]_{K^p_{pr}}\}$ is generated and sent by std_r to pr. The message sent by $std_k \neq std_s$ involved in the communication is the same as $\overline{M}_{std_r}$ with the only difference that $[nonce_s, d_m]_{K^p_{pr}}$ contains a random number rnd. We assume the standalone application to know devices in the proximity of r. Supporting device d_m is then selected based on β by extending the probe request-based mechanism used by user s to start the communication [11]. In particular, the probe request in sender communication phase is extended with the information about the number of devices surrounding the sender of the probe request. Then, r periodically collects and notifies std_r of neighboring devices around it, that is, the ones from which it received a *Probe request* including the number of their neighboring devices. In fact, neighboring devices with less than β devices in their proximity would expose the anonymity of r, whether selected as destination d_m. If this privacy condition is not met, then std_r would simply ask pr to stop the procedure (as for the scenario where c_r is the final destination).

Proxy. Similarly to the previous phase, upon receiving a message $\overline{M}_{std_k}$ sent by std_k, pr decrypts the last two fields of $\overline{M}_{std_k}$ using K^s_{pr} (the private key of pr). The first field contains the identifier id_{cm} of the communication to which $\overline{M}_{std_k}$ belongs, while the second field either d_m and $nonce_2$ in case the decrypted message is the correct one ($\overline{M}_{std_s}$) or a random number rnd otherwise. Upon identifying $\overline{M}_{std_s}$, the proxy waits until at least β messages belonging to the same communication id id_{cm} are received. It then prepares message $\overline{M} = \{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$ and forwards it to d_m, via the cellular operator. Again, waiting for at least β messages and encrypting the whole message with the public key $K^p_{d_r}$ of d_r forbid re-identification by attackers able to observe the cloud and the standalone apps as discussed in Section 5.

Device. Upon receiving message $\overline{M} = \{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$, d_m broadcasts the received message to the nearby devices. Among other devices, r receives the broadcasted message, decrypts it with $K^s_{d_r}$ and SK_r, and reads the file.

4.5. Discussion

The proposed protocol provides an end-to-end anonymity approach for a mobile cloud environment, which supports storage and computation offloading. It allows for a tunable tradeoff between the amount of computation that can be offloaded to the clones in the cloud and the amount of data that are potentially disclosed to the cloud operator. In our protocol, for easy of exposition, we considered one of the extreme scenarios where data in the clone's memory are all stored in the clear (high computation offloading, no confidentiality).

Our protocol employs encryption facilities to hide the two endpoints of a communication according to α and β anonymity preferences. Its behavior can slightly differ from the working discussed in this section depending on α and β. For preference $\alpha = 1$, sender s does not involve Wi-Fi neighbors in its proximity during the sender communication phase, while it directly sends M to c_s via pr. For preference $\beta = 1$, clone c_r directly receives message $\tilde{M}_{pr}$ from pr in the clone-to-clone communication phase, and sends message $\overline{M} = \{[f, nonce_2]_{SK_r}\}_{K^p_{d_r}}$ to r via pr, bypassing d_m, in the receiver communication phase.

Bi-directional communications between s and r can be supported by adding a *response communication* phase to the protocol. This phase can be implemented either as a one-way communication switching s with r or by re-using the anonymous channel created for the communication from s to r. In the latter case, as discussed in [11], involved clones c_i and c_j must be the same for both directions, r must keep track of the identity of c_j, and in turn c_j of the identity of c_i. This can be done by using $nonce_2$ and the knowledge at the proxy.

Finally, there is a subtlety to consider when our anonymous protocol is executed. The file received by r using our protocol is not synchronized with the corresponding clone c_r to avoid sender–receiver re-identification by the cloud provider. If synchronized, in fact, the cloud provider could be able to observe a file stored in c_s that is then stored in c_r. A file received by r can be synchronized with c_r, if and only if the file has been previously modified by r.

5. Security analysis

We assess the security of our protocol against possible adversarial entities aiming to reduce preserved anonymity to (1, 1)-anonymity. In particular, we focus on the novel security features introduced by our proposal and evaluate: (i) The security of our solution against a malicious cloud operator that tampers with the memory of clones (Section 5.1), (ii) the security against malicious clones and standalone apps (Section 5.2), (iii) the security against colluding cloud provider, clones, and standalone apps (Section 5.3). We note that, as far as malicious devices, malicious cellular network operator, and adversary tampering with the proxy are concerned, the security of the scheme proposed in this paper is the same as the one discussed in [11].

5.1. Cloud operator tampering with clones' memory

Adversary and capabilities. We consider an adversarial cloud operator that, beyond eavesdropping and analyzing all the traffic going through his domain, can also tamper with the memory of the clones it hosts.

Execution of the attack. Since clones (e.g., Android virtual machines) are deployed in the physical architecture of the cloud operator, a malicious cloud can indeed inspect the memory of the clones, retrieve cryptographic keys, and decrypt all the communications involving the clone.

Defense. Our proposal is resilient against this attack, for a simple but effective reason: All clones involved in the protocol (i.e., c_s, c_r, as well as the supporting clones) will "blindly" execute a set of operations according to the received messages. Since these operation are, for all the clones involved, "meaningful" operations (e.g., selecting and sending one of the files they store), the cloud operator cannot discern the actual c_s and c_r from the supporting nodes. More specifically, let us consider the *Sender Communication* phase of our protocol, as discussed in Section 4.2. Message M_{pr} received by each clone involved in this step does not require any computation. The clone just needs to select the file f corresponding to id_f and send it (in the *Clone-to-Clone Communication* phase) to the corresponding standalone application. Therefore, in the last step of *Clone-to-Clone Communication* and the first step of *Receiver Communication*, each of the supporting clones acts simply as a forwarder of message $\tilde{M}_{c_j}$, while c_j only decrypts a random number $nonce_2$ in $\tilde{M}_{pr}$.

Result of the attack. Our protocol provides at least (α, β)-anonymity in the worst case.

5.2. Malicious clones and standalone apps

Adversary and capabilities. In this scenario the clones can be honest-but-curios or act in a malicious way by tampering with the protocol. At the same time, the standalone apps can support the corresponding malicious clone, or co-operate with either the sender s (receiver r) to identify the other party involved.

Execution of the attack. Honest-but-curious clones obeys to the protocol, while trying to understand whether they are c_s and c_r. Malicious clones also tamper with the protocol by dropping messages. Malicious standalone app of supporting clones can only retrieve the information about the fact that it is the app of neither the sender nor the receiver of the communication.

On the other side, if the standalone app of the sender s (receiver r) is compromised, the standalone app knows it belongs to s (r).

Defense. Similar to the previous scenario, nor the cloud provider neither honest-but-curios clones involved in a communication channel are able to infer the identity of c_s or c_r, and thus the one of the sender or the receiver. This simply follows by the fact that the knowledge of the clones cannot be bigger than the one of the cloud provider that hosts them. Malicious clones tampering with the protocol by dropping messages do not endanger the anonymity of senders or receivers either—note that, our protocol is general enough to forbid an adversarial clone from understanding its role in the protocol. All this attack could achieve is at most a denial of service—the messages are dropped by either c_s or c_r. However, in this case the corresponding users will eventually detect this behavior, and possibly change cloud provider to mitigate it.

If the standalone app of a malicious clone is also malicious, the user privacy is still preserved. In fact, even in the worst case scenario, when this happens for the sender (receiver) standalone app, the identity of the sender (receiver) is protected by the fact that the malicious *std* does not know the real identity of the corresponding clone. We achieve this by storing random numbers in c_s and c_r of $\tilde{M}$, which are pre-installed in the standalone app without any link to the real identities of the involved parties.

Result of the attack. Our protocol provides at least (α, β)-anonymity in the worst case.

5.3. Colluding cloud provider, clones, and standalone apps

Adversary and capabilities. We consider the possibility of collusion among cloud provider, clones, and standalone apps.

Execution of the attack. The attacker controls the network on the cloud, and either the couple (c_s, std_s), the couple (c_r, std_r), or both.

Defense. The defense against this attack is given by the complexity of the attack itself. The attack might be very costly to be implemented, while it might provide limited results in terms of retrieved information. In fact, it requires to compromise clones and standalone apps of both sender and receiver, and have control of the cloud network (e.g., support by the cloud provider), to access communications involving a single pair of sender and receiver.

Result of the attack. When only one among the couples (c_s, std_s) and (c_r, std_r) is compromised by an attacker also controlling the network in the cloud, our protocol can still guarantee $(1, \beta)$-anonymity when (c_s, std_s) is compromised, and $(\alpha,1)$-anonymity when (c_r, std_r) is compromised. But, if the attacker compromises c_s, c_r, std_s, and std_r at the same time, and have the support of the cloud, it can violate the privacy of both sender and receiver. This is the only case in which the attacker fully identifies both parties in a communication.

We note that the proposed attack is very expensive since standalone apps and cloud clones reside on different platforms — the clones on the cloud, whereas the standalone apps on decoupled machines on the Internet — and requires a supporting cloud provider. Also, a single occurrence of this attack would uncover communications only involving a single pair s and d. Thus, though possible, it is almost impossible for an attacker to simultaneously have a full control of both clones and standalone apps for all possible sender–receiver couples in the system.

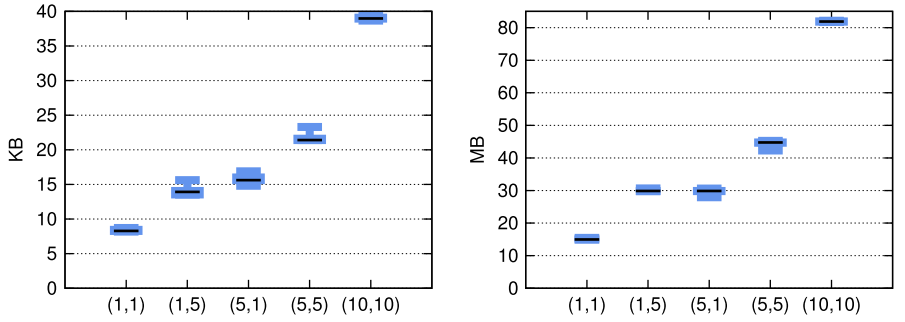
All remaining combinations including an attacker observing the cloud and the standalone apps are not able to achieve (1, 1)-anonymity.

6. Experiments

In this section we investigate on the possible overheads induced by our anonymity protocol. The evaluation focuses on the entities that suffer from hardware-related limits (the battery-limited smartphones), and on the proxy, which could introduce bottlenecks that harm the usability of the system. The protocol is tested for messages with two types of content: A regular text message of 160 Bytes (SMS) and a mp3 file of 3.87 MB (MP3). Each experiment is repeated 30 times and the results are aggregated. To measure the energy-related costs on the phone side we used the Power Monitor[1] meter. It samples the smartphone battery with high frequency (i.e., 5000 Hz) so to yield accurate results on the battery power, current, and voltage. The mobile devices in our testbed were Samsung Galaxy S+ devices, 1.4 GHz Scorpion CPU, and 512 MB of RAM running Android 2.3. The proxy and the clones were running on a commodity laptop with the following characteristics: Ubuntu 14.04, Intel Core i7-4500U CPU, 1.80 GHzX4, 8 GB RAM. Algorithm AES with 192 bit key length was used for symmetric encryption and RSA with 1024 bit key length for asymmetric encryption.[2]

[1] https://www.msoon.com/LabEquipment/PowerMonitor/.

[2] We note that, for convenience, we used 1024 bit length for asymmetric encryption though the latest NIST recommendations suggest using a 2048 bit long RSA key (http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800–57Pt3r1.pdf). This choice does not affect our experimental results as the RSA key is mostly used to encrypt the 192 bit long symmetric key in secret session communications.



(a) SMS overhead. (b) MP3 overhead.

Fig. 4. Traffic overhead per message varying (α, β) anonymity preferences. The graphics include the max, min, and quartiles values.

6.1. Evaluation on the proxy-side

The proxy plays a crucial role in the system and its anonymity: It is responsible of "coupling" device traffic towards clones and standard applications and vice versa. As such, it is important to study the amount of traffic per message the proxy needs to handle during a single one-to-one communication among devices. Recall that the proxy is involved in all the three steps of the protocol, while the traffic overhead is determined by the number of clones (standalone apps) involved in a single communication. Indeed, the proxy needs to receive as many messages as clones (standalone apps) both in the clone-to-clone and receiver communication steps, from which it discriminates the correct message to push forward in the protocol (see Figs. 2 and 3). This number is strictly related to the (α, β) anonymity preferences of the communication: There are at least α clones (standalone apps) involved in the clone-to-clone step, and at least β clones (standalone apps) involved in the receiver communication step. For this reason, we have studied the traffic handled by the proxy varying α and β in the set $\{1, 5, 10\}$. The corresponding results are shown in Fig. 4. As one might expect, the traffic handled by the proxy is higher for higher values of α and β, for both types of content exchanged among devices. What is surprising, however, is that the amount of traffic does not grow in a proportional way w.r.t. the anonymity parameters. This observation indicates that higher anonymity guarantees can be met by our protocol without inducing severe traffic overheads to the proxy. Recall that in our testbed the proxy runs on a commodity laptop. Nonetheless, we believe that in real deployments the proxy could be efficiently implemented and deployed on a distributed set of high-performing servers, which will boost its performance and that of the overall protocol.

6.2. Evaluation on the device-side

The anonymity protocol involves costly encryption/decryption operations as well as sending message bundles that include the file index to be transmitted and other data necessary to guarantee the anonymity of the communication. In this section we discuss these costs from the perspective of the devices and compared them with the ones of a plain email protocol. Although the email protocol does not involve the cloud and does not guarantee any anonymity properties to users, it served as a benchmark in our evaluation.

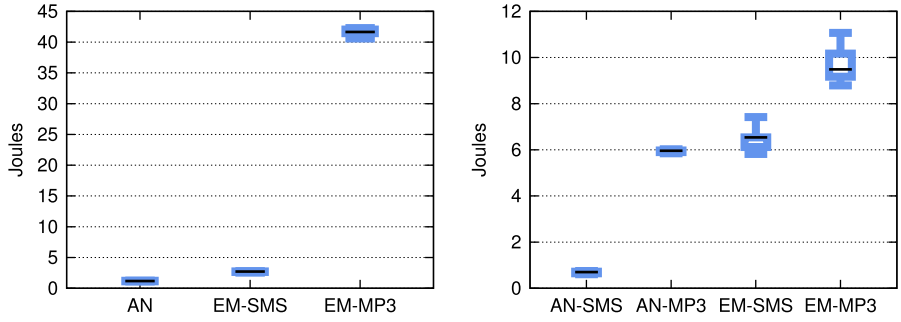
6.2.1. Overhead on sender device

We start with the energetic costs on the sender device. They include the costs of (i) the generation of the bundle M to be forwarded to the next hop by short ad hoc links (sender communication step) and (ii) the communication through WiFi direct. We note that these costs are content-independent. Indeed, according to our protocol, the content is already on the cloud, and only the *id* of the corresponding file is sent within the message bundle to identify the corresponding file within the cloud and forward it anonymously towards destination. The results are presented in Fig. 5(a). It is clear how, despite the several cryptographic operations involved, the energetic overhead on the source-side is less than 1.25 J.[3] When compared to the plain email protocol the sender spends 2 times less for short messages (comparable to SMS) and up to around 20 times less for larger content (mp3 file).

6.2.2. Overhead on receiver device

Now let us consider the costs on the receiver side. Again, they include the energy spent for receiving the message bundle by d_m (receiver communication step), and for decrypting the bundle to finally read the content. We note that, in this case, the file is included in the bundle. This makes the costs dependent on the type of content that is being sent. The results are presented in Fig. 5(b) and show that the consumption of our anonymity protocol is again considerably lower than that of the

[3] When fully charged, the capacity of the battery of the devices involved in the testbed contains around 22 kJ of energy.



(a) Energetic overhead on the source device: Anonymity protocol (AN) vs E-mail (EM).

(b) Energetic overhead on the receiving device: Anonymity protocol (AN) vs E-mail (EM).

Fig. 5. Energetic overhead on the source and destination devices. The graphics include the max, min, and quartiles values.

plain email protocol. In particular, it results 0.78 J for the short text case and around 6 J for the mp3 audio file. Considering the capacity of almost 22 kJ of the battery of the involved devices, these values are particularly low. Most importantly, when compared to the plain email protocol the consumption is 3.5 times lower for the short text case, and around 2 times lower for the mp3 file. Our investigation showed that this difference is mostly due to the considerably longer download time of the email content, which is certainly dependent on the mailing server. This forces the destination device to keep its communication interface up for a longer time, which induces considerably more energy consumption. Differently, in our protocol, the communication is ad hoc between the receiving device and d_m. The communication link exploits the WiFi direct protocol for device-to-device communication, which results more efficient from the receiving device's perspective.

6.2.3. Overhead on relay devices

The anonymity protocol involves also other devices—those that behave as relays through ad hoc links on both the sender and the receiver communication steps of the protocol. The devices involved in the sender communication step, however, have a much easier job than those involved in the latter. Indeed, they only need to forward the bundle M generated by the source a step further. According to our experiments, the energetic cost is less than 1 J. Differently, in the receiver communication step, we distinguish two types of devices: The d_m, in charge of broadcasting the message bundle to all β devices in its proximity, and a given device d_x which is not the destination of the message, but does not know it yet. It is clear that the cost induced to d_x is similar to that of the receiver. However, the cost of d_m is dependent on the parameter β of the protocol, which determines the number of WiFi-direct transmissions d_m needs to perform. According to our experiments, d_m will spend 1.8 J, 8.9 J, and 17.8 J for $\beta = 1$, $\beta = 5$, and $\beta = 10$. Again, these values are very low w.r.t. the 22 kJ battery capacity of the devices involved in the testbed.

7. Conclusions

We presented a protocol for anonymous end-to-end communications among users in a mobile cloud environment, where the cloud clones handle part of the communication towards destination. The attack model considered is unprecedented. It includes devices, network operators, and the cloud provider behaving as malicious entities, and the possibility of all of them to collude. In this scenario, we built a delay-tolerant solution that provably guarantees (α, β)-anonymity, and evaluated its performance on a real-life testbed. Our future work will extend our approach to scenarios where exchanged files may contain information on sender/receiver, will depart from the assumption of having a standalone app available for each user in the Internet, and will provide a formal security analysis of our protocol using automatic cryptographic protocol verifiers, such as ProVerif.

References

[1] B.G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: EuroSys'11.
[2] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: MobiSys'10.
[3] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: IEEE INFOCOM'12.
[4] M.V. Barbera, S. Kosta, A. Mei, V.C. Perta, J. Stefa, Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system, in: IEEE INFOCOM'14.
[5] M.V. Barbera, S. Kosta, J. Stefa, P. Hui, A. Mei, CloudShield: Efficient anti-malware smartphone patching with a P2P network on the cloud, in: IEEE P2P'12.
[6] G. Portokalidis, P. Homburg, K. Anagnostakis, H. Bos, Paranoid android: versatile protection for smartphones, in: ACSAC'10.
[7] M.V. Barbera, S. Kosta, A. Mei, J. Stefa, To offload or not to offload? The bandwidth and energy costs of mobile cloud computing, in: IEEE INFOCOM'13.
[8] S. Kosta, C. Perta, J. Stefa, P. Hui, A. Mei, Clone2Clone (C2C): Peer-to-Peer Networking of Smartphones on the Cloud, in: HotCloud'13.
[9] S. Kosta, V.C. Perta, J. Stefa, P. Hui, A. Mei, CloneDoc: Exploiting the Cloud to Leverage Secure Group Collaboration Mechanisms for Smartphones, in: IEEE INFOCOM'13.

[10] M. Van Dijk, A. Juels, On the impossibility of cryptography alone for privacy-preserving cloud computing, in: USENIX HotSec'10.
[11] C. Ardagna, M. Conti, M. Leone, J. Stefa, An anonymous end-to-end communication protocol for mobile cloud environments, IEEE TSC 7 (3) (2013).
[12] M.V. Barbera, S. Kosta, A. Mei, V.C. Perta, J. Stefa, CDroid: Towards a Cloud-Integrated Mobile Operating System, in: IEEE INFOCOM'13.
[13] C. Ardagna, S. Jajodia, P. Samarati, A. Stavrou, Providing users' anonymity in mobile hybrid networks, ACM TOIT 12 (2013) 1–33.
[14] P. Mittal, M. Wright, N. Borisov, Pisces: Anonymous communication using social networks, 2012, arXiv:1208.6326.
[15] A. Mohaisen, Y. Kim, Dynamix: anonymity on dynamic social structures, in: ASIACCS'13.
[16] A. Mohaisen, H. Tran, A. Chandra, Y. Kim, Trustworthy distributed computing on social networks, in: ACM ASIACCS'13.
[17] K. Puttaswamy, A. Sala, O. Egecioglu, B. Zhao, Rome: Performance and anonymity using route meshes, in: IEEE INFOCOM'09.
[18] K. Puttaswamy, A. Sala, B. Zhao, Starclique: guaranteeing user privacy in social networks against intersection attacks, in: CoNEXT'09.
[19] S. Seys, B. Preneel, ARM: anonymous routing protocol for mobile ad hoc networks, Int. J. Wirel. Mob. Comput. 3 (2009) 145–155.
[20] Y. Zhang, W. Liu, W. Lou, Y. Fang, MASK: Anonymous on-demand routing in mobile ad hoc networks, IEEE TWC 21 (2006) 2376–2385.
[21] L. Chaum, Untraceable electronic mail, return addresses, and digital pseudonyms, CACM 24 (1981) 84–90.
[22] R. Dingledine, N. Mathewson, P. Syverson, Tor: The second–generation onion router, in: USENIX Security'04.
[23] M. Senftleben, M. Bucicoiu, E. Tews, F. Armknecht, S. Katzenbeisser, A.-R. Sadeghi, Mop-2-mop—mobile private microblogging, in: Financial Cryptography and Data Security, Vol. 8437, 2014, pp. 384–396.
[24] J. Daubert, L. Bock, P. Kikirasy, M. Muhlhauser, M. Fischer, Twitterize: Anonymous micro-blogging, in: AICCSA'14.
[25] M. Alizadeh, S. Abolfazli, M. Zamani, S. Baharun, K. Sakurai, Authentication in mobile cloud computing: A survey, J. Netw. Comput. Appl. 61 (2015) 59–80.