Secure Trust Management for the Android Platform

Raimund K. Ege

Dept. of Computer Science
Northern Illinois University
DeKalb, IL, USA
ege@niu.edu

Abstract—**Smartphones change the way we use the Internet. No longer are we limited to media consumption, but participation in Social Media etc. allows us all to become media producers. Moreover, with its computing power and network connectivity, the smartphone can become a peer in a peer-to-peer based content delivery network. Securing property rights to the media must be part of such sharing and propagation. This paper explores the capabilities available to the Android platform to secure such participation, and it describes an architecture for adding trust management to the exchange of media to and from a smartphone user.**

Keywords-Android; security; trust; management; peer-to-peer systems; multi-media content delivery

I. Introduction

Personal digital assistants (PDA) have grown up into Smartphones: with computing power, display and recording capabilities, and – foremost – with broadband internet connectivity. No longer is a phone user limited to making phone calls and reading email, but the user can participate in a host of social applications that are rich in multimedia exchange. The social media scene is full of sites such as Facebook, YouTube, Instagram, Vimeo, etc. Participating in such venues requires that user reveal, typically via a user registration, their identity. In addition some proof might be required to authenticate the identity. Most sites, however, are satisfied once an email address is verified. The registration serves the purpose of adding a layer of trust to the consumption but also to the submission of media to these sites.

The next step in the evolution of the smartphone is to not just consider it a client to such social media sites, but to let it become an active player in the delivery of the media. The computing power and network connectivity enable the provision of peer-to-peer (P2P) content delivery networks: rather than just down- or up-loading media to one site, media can be shared in such P2P network at a much higher throughput, i.e. no single source bottleneck, and without central control, i.e. big brother registration. The aim of our research is to allow the forming of very large P2P content sharing networks, without central control, but with provisions that instill a degree of trust into the participants.

This paper describes an architecture for adding trust management to the exchange of media to and from a smartphone user. Section 2 gives some background on access control, identity and trust management and relates out work to current research. Section 3 surveys which elements of security to ensure confidentiality, integrity and availability are available to mobile platforms, with a specific focus on what is available to Android smartphones. Section 4 elaborates on how our approach defines and gauges trust, and how such trust is maintained, secured and shared in a central-server-less P2P environment. Section 5 outlines our prototype implementation with Java peers, including peers running on Android smartphones. The paper concludes with some lessons we learned and our future perspective.

II. Background

Much research has been conducted on access control and digital rights management. Access control is common place in many applications. A server maintains a database of user and account information. A user gains access to the system by providing a user id with additional security information, typically a password. Once authenticated, the user is “trusted”, i.e. is allowed to participate in the system’s mission. The information stored by the server can include the users past history of participation, which in turn can be used to augment the level of trust in the user. Other users might contribute to the trust evaluation by submitting feedback on others. The level of trust might determine the level of participation a user is allowed, e.g. users with a low level of trust might be able to consume content, while users with a high level of trust might be able to contribute media.

While central access control makes sense for central systems, systems that don’t have a central point of service, typically out-source their authentication aspect to other players: OpenID [1] is an example: OpenID providers maintain identity information and allow users to choose to associate information with their OpenID that can be shared with the media sites they visit. With OpenID, a password is only given to the identity provider, and that provider then confirms the identity.

In a peer-to-peer system peers need to collaborate and obtain services within an environment that is unfamiliar or even hostile. Therefore, peers have to manage the risks involved in the collaboration when prior experience and knowledge about each other are incomplete. One way to address this uncertainty is to develop and establish trust among peers. Trust can be built either via a trusted third party [2] or by community-based feedback from past experiences [3] in a self-regulating system. Other approaches reported in the literature use different access control models [4] [5] that qualify and determine authorization based on permissions defined for peers.

In such a complex and collaborative world, a peer can benefit and protect itself only if it can respond to new peers and enforce access control by assigning proper privileges to new peers. Trust management can help minimize risk and ensure the network activity of benign entities in distributed systems [6].

Digital rights management has been the focus of many secure content delivery systems. Peer-to-peer and mobile schemes have been introduced. One such effort, OMA DRM [7] – undertaken by the Open Mobile Alliance, an industry consortium – provides a standard framework to secure media for mobile devices. It uses public key infrastructure (PKI [8]) style certificates and public/private key pairs to protect media. Our approach goes further in that it does not require certainty of access right, but rather allows building of graduated trust which enables graduated access control to digital media.

In our prior work [9] we started to develop an understanding on how trust can be quantified, especially when related to the potential reward garnered by a peer who participates in a peer-to-peer content delivery network. In this paper we focus on how to create and maintain trust in a distributed fashion, and how to secure it in a mobile environment.

III. Elements of Mobile Security

Security concepts include confidentiality, integrity and availability. All three of these basic tenants of computer security are essential to our goal of securing trust information in a mobile environment. Via confidentiality we ensure that the communication among peers is only observable to authorized peers. Via integrity we ensure that communication as well as the history of communication is maintained without improper alteration. Via availability we insure that peers can readily join the content delivery network and that their trust values and histories are available in making decisions on their degree of participation.

We ensure confidentiality via encryption. Today’s smartphones have enough computing power to handle encryption: asymmetric encryption, e.g. the Diffie-Hellman key exchange protocol, can be used to establish session keys that ensure the confidentiality as data, such as identity and trust information, and media streams are exchanged. Standard block cyphers, e.g. DES or AES, are used to encrypt sensitive data, and standard stream cyphers, e.g. RC4, secure media streams. We ensure integrity via digital signatures. Again, modern smartphones can handle standard algorithms such as DSA, etc. Key management is also standardized and it is quite common for a smartphone to maintain a local key store on the device itself.

Our prototype implementation is done entirely using the Java programming language. The Java platform strongly emphasizes security, including language safety, cryptography, public key infrastructure, authentication, secure communication, and access control. The Java Cryptographic Architecture [10] defines a "provider" architecture. Multiple providers are available in a typical Java development environment. We choose the “Bouncy Castle” [11] Java implementation, which is widely available, including for the Android platform. On Android we are actually using the “Spongy Castle” [12] variant that replaces the standard (but older) Bouncy Castle version for Android which is provided by Google.

In summary, all elements of a public key infrastructure (PKI) are readily accessible to any peer in a peer-to-peer content delivery network, even to a mobile ad-hoc peer via a smartphone.

IV. Trust Model

Peer-to-peer is a communications model in which each party has the same capabilities and either party can initiate a communication session. Each party can become a peer. Once a peer is identified, it is a matter of trust whether and to what degree the peer is allowed to partake in the shared media content. We start by assigning each peer a numeric trust value in the range of -1 to 1, where 0 represent a neutral value, i.e. the network as a whole does not have a judgment on the trustworthiness of the peer. A positive value reflects more trust, a negative value reflects mistrust. As a peer participates in the network, i.e. is part of the “swarm”, each of the peer’s transactions is judged and results in an update to the trust value. At +1, a peer is considered trustworthy enough to partake in the shared trust management of the swarm. At -1, the peer is disqualified from any further participation in the swarm.

The trust value and the peer’s history of relevant transactions are maintained in a container we call “trust nugget”. This nugget contains detailed information on a peer’s participation, such as length and quality of stream transmission, ratio of seed vs. leech behavior, judgments of other stream participants, etc. The nugget content is signed with a special master private key. It can be verified only via the special master public key. This ensures that the trust information maintains its integrity, even as it is shared with peers in the swarm that have lower trust values.

Trust information per peer is maintained by trusted peers, i.e. peers with trust values greater than 1. The sole requirement for starting a new swarm is the existence of an initial trusted peer that we call the “boot strap peer”. This peer initially creates the master public/private key pair that is only shared with other trusted peers. A trusted peer maintains a database of trust nuggets for all peers in the swarm. Again, initially, only one peer, i.e. the boot strap peer, has such a database, but as new peer attains trusted peer status, it receives the database, and also participates in synchronizing the database among all trusted peers.

The trust value for a peer is computed from the peer's history of transactions. The computation is done by a trusted peer whenever a peer reports on another peer. A common scenario is that a peer serves as a source of media content: it makes the content available to the peers in the swarm. Once a peer has consumed the content, the "source" peer notifies a trusted peer of the peer's behavior: good or bad. The trusted peer enters a new transaction into the peer's nugget and signs it with the master private key.

When a peer acts as a source peer, i.e. it makes new content available to the swarm; it can set a trust threshold, i.e. a minimum trust value, required for any peer to access the content. Only peers whose trust value meets the threshold can participate. The source peer also determines the weight of a peer's participation when computing a peer's new trust value.

Trusted peers are the backbone of our trust model. New peers need to register with one trusted peer which creates a trust nugget for the new peer. The new peer also creates a public/private key pair and submits its public key to the trusted peer. Other components of the model are the provider of the original source data, i.e. a "source peer", and peers that consume the multimedia content. Peers can also serve as further sources in a peer-to-peer download model.

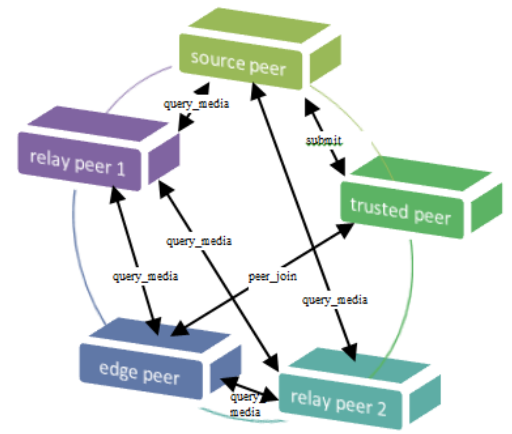


Figure 1: peer-to-peer network

Figure 1 shows an example snapshot of a content de-livery network with one source peer, one trusted peer, and 3 regular peers: 2 relay peers and one edge peer. The source peer is where the content data is produced, en-coded, encrypted and made available. The source submits the stream info to a trusted peer. Peers connect to a trusted peer for authentication and to receive the download credentials. A peer that only downloads is called an "edge peer". Once the peer starts serving the stream to other peers, it becomes a "relay peer".

All peers together maintain a peer group, i.e. information on which peers are actively part of the content delivery network. The trusted peer initially informs the peers in the peer group which source peer to download from: peer 1 is fed directly from the source peer; peer 2 joined somewhat later and is now being served from the source peer and peer 1; the edge peer joined last and is being served from peer 1 and peer 2. In this example, peer 1 and 2 started out as edge peer, but became relay peers once they had enough data to start serving as intermediaries on the delivery path from original source to ultimate consumer.

V. JAVA IMPLEMENTATION

Our implementation has 4 major components:

(1) A set of trusted peers, initially just one: the boot strap peer;

(2) An application that allows a source peer to submit information about a content stream;

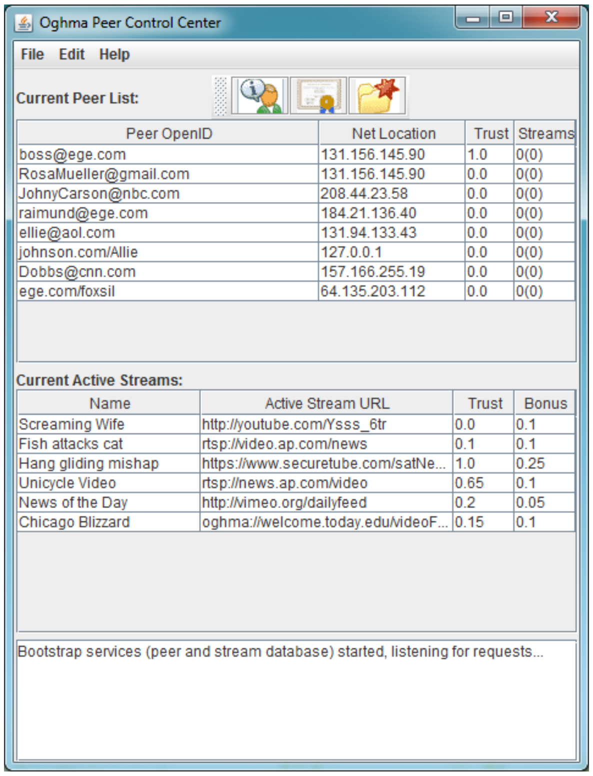
(3) A relay peer that consumes data, e.g. shows the video, and makes it available to other peers; and

(4) An edge peer to run on an Android mobile device. Android is implemented in Java and therefore offers a flexible and standard set of communication and security features.

A. Trusted Peer

The central component of our architecture is the trusted peer. It maintains a database of all peers and a tracks the collection of data streams that are made available by sources. Our trusted peer prototype presents a display of all peers and streams (see Figure 2).

When a new peer connects to a trusted peer, authentication is achieved via the peer's openID, which is validated the openID provider. If the peer is new, i.e. the trusted peer has no trust nugget for the peer, the new peer must provide its public key and a new trust nugget is created. The peer's public key is later provided to source peers who will use it to encrypt content destined for that peer. "ellie@aol.com" could have been the result of the peer leaking parts of the stream to non-authorized parties.



Current Peer List:

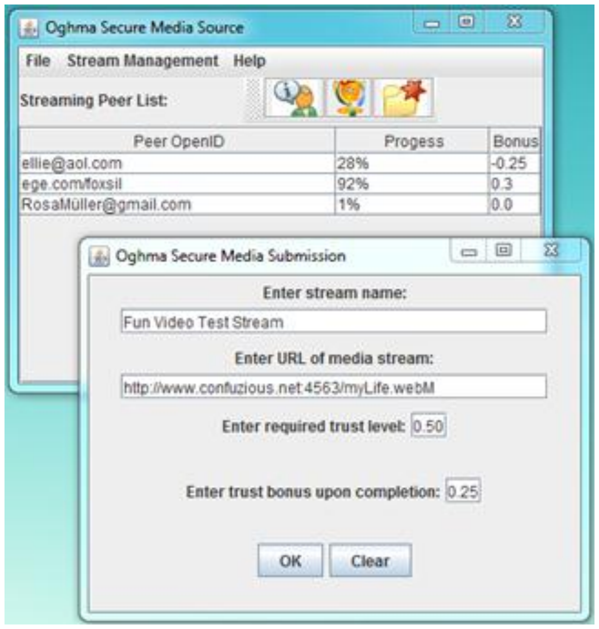
Peer OpenID	Net Location	Trust	Streams
boss@ege.com	131.156.145.90	1.0	0(0)
RosaMueller@gmail.com	131.156.145.90	0.0	0(0)
JohnyCarson@nbc.com	208.44.23.58	0.0	0(0)
raimund@ege.com	184.21.136.40	0.0	0(0)
ellie@aol.com	131.94.133.43	0.0	0(0)
johnson.com/Allie	127.0.0.1	0.0	0(0)
Dobbs@cnn.com	157.166.255.19	0.0	0(0)
ege.com/foxsil	64.135.203.112	0.0	0(0)

Current Active Streams:

Name	Active Stream URL	Trust	Bonus
Screaming Wife	http://youtube.com/Ysss_6tr	0.0	0.1
Fish attacks cat	rtsp://video.ap.com/news	0.1	0.1
Hang gliding mishap	https://www.securetube.com/satNe...	1.0	0.25
Unicycle Video	rtsp://news.ap.com/video	0.65	0.1
News of the Day	http://vimeo.org/dailyfeed	0.2	0.05
Chicago Blizzard	oghma://welcome.today.edu/videoF...	0.15	0.1

```
Bootstrap services (peer and stream database) started, listening for requests...
```

Figure 2: Trusted Peer Prototype


Oghma Secure Media Source
File Stream Management Help

Streaming Peer List:

Peer OpenID	Progess	Bonus
ellie@aol.com	28%	-0.25
ege.com/foxsil	92%	0.3
RosaMüller@gmail.com	1%	0.0

Oghma Secure Media Submission

Enter stream name: Fun Video Test Stream

Enter URL of media stream: http://www.confuzious.net:4563/myLife.webM

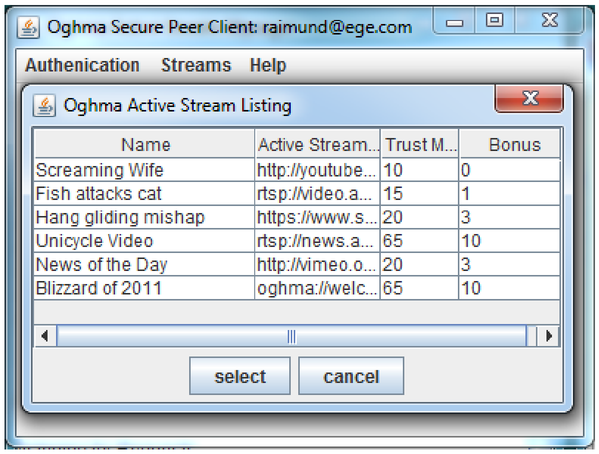
Enter required trust level: 0.50

Enter trust bonus upon completion: 0.25

Figure 3: Media Source Prototype

B. *Relay Peer*

The relay peer application is used to allow a peer to authenticate with a trusted peer, get a listing of available streams, make a selection, display the stream and finally also make the stream available to other peers downstream. Figure 4 shows a screen capture of the Java Peer Client prototype:


Oghma Secure Peer Client: raimund@ege.com
Authentication Streams Help
Oghma Active Stream Listing

Name	Active Stream...	Trust M...	Bonus
Screaming Wife	http://youtube...	10	0
Fish attacks cat	rtsp://video.a...	15	1
Hang gliding mishap	https://www.s...	20	3
Unicycle Video	rtsp://news.a...	65	10
News of the Day	http://vimeo.o...	20	3
Blizzard of 2011	oghma://welc...	65	10

Figure 4: Relay Peer Prototype

Once the peer is authenticated with a trusted peer, it can request a list of available streams. Figure 4 shows all streams that are currently available with their name, required trust value, and potential bonus. The peer can make any selection. The reason why all stream are displayed, even the ones which require a higher trust value than what the peer currently has, is to give the peer an incentive to first participate in another stream to add the bonus to its trust value. However, only streams can actually be selected for which the peer is currently qualified.

Once the peer has selected a stream for viewing, the trusted peer will transmit the necessary information to enable the peer to start download. It will get the set of all locations at which the media stream is available. The peer then contacts the source locations at their streams' URLs and starts downloading content data, i.e. the sequential frames of the video stream.

In general, peers can do 3 things:

(1) they continuously request frames from other peers (the original source is viewed as just another peer) and store them;

(2) they may display the frames as video to the user of the peer device;

(3) and they make the stored frames available to other peers. Figure 5 shows our prototype Java implementation of our Peer Client while it displays the requested video:

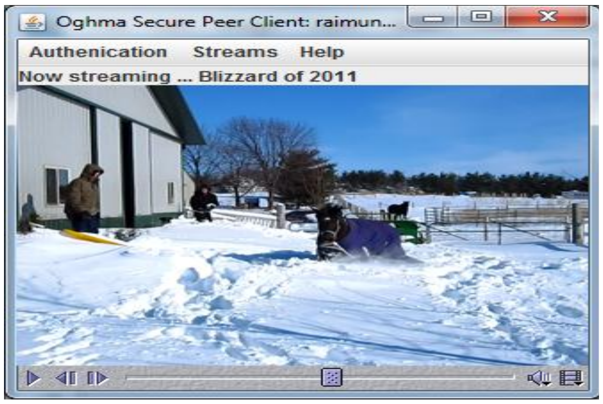

Oghma Secure Peer Client: raimun...
Authenication Streams Help
Now streaming ... Blizzard of 2011

Figure 5: Relay Peer Prototype

Peers don't need to provide all 3 services. A peer that provides only service (1) and (2) is an "edge" peer, i.e., an end user consumer. A peer that provides service (1) and (3) is a "relay" peer. Relay peers are specifically important for peers that have limited access to the public Internet, i.e., peers behind network boundaries, such as a NAT firewall. In addition, peers stay in contact with each other to continuously update the peer group and source data availability.

C. Edge Peer

The final component of our prototype framework is our proof-of-concept edge peer implementation for the Android platform. Figure 6 shows three screens: "login", "stream selection", and "stream play" of our Android prototype edge peer application.

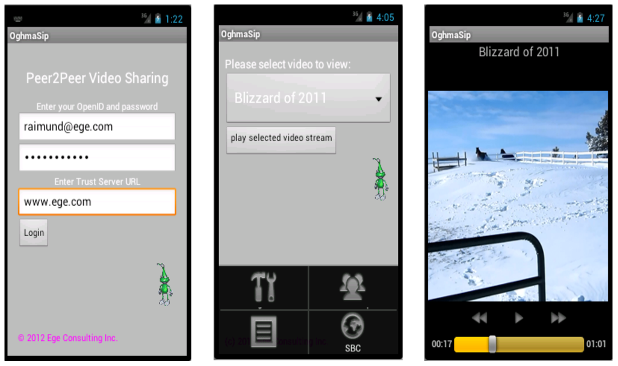


Figure 6: Android Prototype App

First, each peer is authenticated with its OpenID credentials. The user enters userid and password, plus the URL of a boot strap trusted peer. If the peer is new to the content delivery network, it will also generate a public/private pair of Diffie-Hellman keys, keep one private and submit the public one to the trusted peer. Once authentication is achieved, i.e. the OpenID provider has sent the authorization token, the user is shown which streams are currently available on the next screen. Once the "play selected video stream" button is pressed, and a sufficient read-ahead buffer has been accumulated, the video stream starts playing on the Android device.

CONCLUSION

In this article, we described an architecture for peer-to-peer based content delivery networks that empowers participating peers. Peers joining peer groups and establish trust among each other. Benevolent participation, i.e., consuming, producing, sharing and propagating media content, increases the understanding of shared trust. The trust information, i.e., the history of p2p transactions, is maintained in secure manner, signed with the private key of a trusted peer. Trust can also be lost; each unsuccessful transaction lowers the peer's trust value, ultimately to the point where the peer is ejected from the peer group.

We also described a prototype implementation written in Java to boot strap a P2P network, and includes a Java-based client for the Android platform for smartphones. Our intention was to demonstrate that the security capabilities of the Java Cryptographic Architecture are sufficient, and that its provider implementations run well on Android smartphones.

Our next steps will be to orchestrate and simulate a large actual swarm, i.e. an actual network with a large number of participating peers. Our goal is to measure how robust our trust management is and how well it withstands the introduction of malevolent peers.

REFERENCES

[1] OpenID, http://www.openid.net. [accessed September 13, 2012]

[2] J Y. Atif. Building trust in E-commerce. IEEE Internet Computing, 6(1):18–24, 2002.

[3] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. Communications of the ACM, 43(12):45–48, 2000.

[4] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. In SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies, pages 41–52, New York, NY, USA, 2001.

[5] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. ACM Transaction Database System, 26(2):214–260, 2001.

[6] H. Li and M. Singhal. Trust Management in Distributed Systems. Computer, vol. 40, no. 2, pp. 45-53, Feb. 2007.

[7] OMA Digital Rights Management V2.0, http://www.openmobilealliance.org/technical/release_program/drm_v2_0.aspx. [accessed September 20, 2012]

[8] [10] C. Adams and S. Lloyd. Understanding PKI: concepts, standards, and deployment considerations. Addison-Wesley Professional. ISBN 978-0-672-32391-1. 2003.

[9] Raimund K. Ege. OghmaSip: Peer-to-Peer Multimedia for Mobile Devices. The First International Conference on Mobile Services, Resources, and Users (MOBILITY 2011), pages 1-6, Barcelona, Spain, October 2011.

[10] Java Cryptography Architecture (JCA) Reference Guide. http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html. [accessed September 20, 2012]

[11] The Legion of the Bouncy Castle. http://www.bouncycastle.org/java.html. [accessed September 20, 2012]

[12] Spongy Castle. http://rtyley.github.com/spongycastle. [accessed September 20, 2012]