JOURNAL OF INFORMATION, KNOWLEDGE AND RESEARCH IN COMPUTER ENGINEERING (ISSN 0975 – 6760, GIF: 01.1135, SIF: 02.502)

Secure Instant Messaging Architecture for Mobile Devices

Ms. Anagha Kapse
Raisoni Group of Institution
HR department
Nagpur, Maharashtra.

Mr. Shailesh Patre
Raisoni Polytechnic College
Nagpur, Maharashtra.

***Abstract*—We propose a secure instant messaging architecture which is equipped with a self-message-destructing feature for sensitive personal information applications in a mobile environment. We design Message Encryption and Exchange Scheme and the format of a self-decryption message to show how to exchange these messages. The instant Messaging-based system enables senders to set time, frequency, and location constraints. These conditions determine when the transmitted messages should be destructed and thus become unreadable for receivers. The instant Messaging-based system securely sends self-destructible messages to receivers in a way that it uses ephemeral keys to encrypt the messages and transmits the encrypted messages to the designated receivers via instant messaging service in real time. We have implemented a simple messenger application on the Android platform and have evaluated its performance to show that our proposed instant Messaging architecture is practical and feasible for sensitive personal information communication on mobile devices.**

I. INTRODUCTION

The popularity of instant messaging has grown in an exponential way. However, most popular instant messaging services often trade speed for security. Thus, the users are subjected to constant eavesdropping, unwanted leakage of confidential information, and even compromises of users' media used in instant messaging. There are several secure instant messaging services for personal or enterprise use. The server may contain the unencrypted messages for administrative purposes. Thus, the user cannot guarantee the confidentiality of the message delivered. The unwanted consequence of leaving unencrypted messages on servers may cause leakage of personal information, or even identity theft. For example, the commercial instant messaging.
providers may choose to mine these plaintext messages for target advertisements.

To solve the aforementioned issues, Kikuchi, *et al* [3] presented a secure instant messaging protocol preserving confidentiality against administrator. A limitation of their protocol is that it needs to modify the instant messaging server to satisfy the protocol requirements.

In addition to encrypting the messages for confidentiality, there are some systems with a "self-data-destructing" feature. These systems focus on making data disappear after a pre-specified time. In other words, data is encrypted and the encryption key is deleted after the expiration time, so that the encrypted data becomes unrecoverable, i.e., self-destructed. However, none of integrate this concept with the instant messaging service using mobile devices and take account of their limitation.

We present a secure instant message encryption and exchange scheme, which can be used directly atop an existing instant messaging service architecture, so that it is easily deployable, leveraging the existing infrastructure. Specifically, we add a self-message-destructing feature to get the enhanced secure instant messaging architecture,The scope of our mechanism is not limited to instant text messaging. In fact we can extend its usage to enhance the security of transmission channels, such as message push and synchronization technologies among devices connected in the Cloud. To demonstrate the feasibility of our proposed instant Messaging architecture, we have implemented a simple messenger application using XMPP (Extensible Messaging and Presence Protocol) instant messaging on the Android platform.

A. Contributions

This paper introduces the instant Messaging, an enhanced secure instant messaging architecture with a self-message-destructing feature for sensitive personal

Information applications in a mobile environment. The primary concepts are listed as follows.

A destructible message encryption and exchange scheme atop the existing instant messaging services to make the message self-destructible. Our approach does not rely on external, special-purpose dedicated services.

An implementation on Android to demonstrate instant Messaging-based mobile app and evaluate its benchmark performance in terms of key generation and message encryption and decryption time.

B. Instant MESSAGING ARCHITECTURE

In this section, we discuss instant Messaging, an enhanced secure instant messaging architecture with a self-message-destructing feature. The Instant Messaging-based system is for handling the sensitive data encryption and transmission problems in an insecure mobile environment, and making sure the encrypted data is unrecoverable for any one once the destructible message's constraints are satisfied. Its components include: sender, receiver, Instant Messaging Server, and an optional Ephemeral Public Key Manager which is used by the receiver to delegate their ephemeral public keys. We give below the details of each component.

A. Sender

The sender acts as an instant messaging client. When a sender would like to send a receiver the message containing sensitive personal information, the Instant Messaging-based application installed in sender‘s mobile device must be able to do the following tasks: (1) generate a long-term public and private key pair; (2) obtain an ephemeral public key from the receiver via an instant messaging service (IMS) and assign the constraint governing the condition to delete this ephemeral key; (3) generate the encrypted message using this ephemeral public key; (4) send the destructible encrypted message, including the ciphertext encrypted with a secret session key, the related encryption information encrypted with this ephemeral public key, and this ephemeral public key's identifier, to the receiver via an IMS;

(5) provide the ephemeral public key when requested by the receiver via an IMS.

B. Receiver

The receiver also acts as an instant messaging client. To receive the destructible messages, the Instant Messaging-based application installed in receiver‘s mobile device must be able to do the following tasks: (1) generate a long-term public and private key pair; (2) generate a series of ephemeral key pairs; (3) receive the destructible encrypted message from the sender via an instant messaging service (IMS); (4) decrypt the destructible encrypted message; (5) provide the ephemeral public key when requested by the sender via an IMS; (6) show the sensitive personal information plaintext on screen after decryption without storing the plaintext in stable storage; (7) securely delete the ephemeral private key (i.e., overwrite it with random data) when the destructible encrypted message's constraints are satisfied.

C. Instant Messaging Server

An instance messaging server is a communication server which includes the basic features such as the authentication of the user accounts, the management of the user's presence status, and the instant message transmission between users. It may also support the enhanced features, such as the encryptions of the authentication process and the transmission channel, to protect the authentication information and the confidentiality of messages respectively.

D. Ephemeral Public Key Manager

Ephemeral Public Key Manager is an instance messaging agent which runs on a server to keep itself available for senders and receivers, and supports ephemeral public keys management. It is needed if we intend to support off-line messaging that allows a sender to send the messages to an off-line receiver. Receivers can delegate their ephemeral public keys to the Ephemeral Public Key Manager's database prior to getting off-line, so that senders can still get the required ephemeral keys from the Ephemeral Public Key Manager, rather than awaiting receivers to become on-line later. The key manager is an important role to support the message push and synchronization technologies among mobile devices connected in the Cloud because mobile devices are not always online.

III. DESTRUCTIBLE MESSAGE ENCRYPTION AND EXCHANGE SCHEMES

Our schemes are designed for the mobile devices, so some limits of the mobile devices, e.g. availability of the network connection and computing power, are under our consideration deliberately. Fig.1 shows a basic encryption and exchange scheme under the situation that the receiver is on-line. Then we extend our basic scheme to support the situation that the receiver is off-line in Fig. 2.

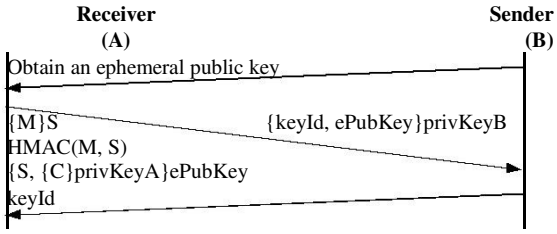

Figure 1. Basic scheme: send messages when the receiver is on-line.

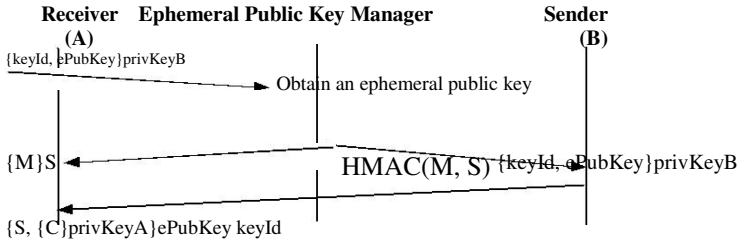

Figure 2. Enhanced scheme: send messages when the receiver is off-line.

A. *Basic Scheme*

1) Setup: A generates his long-term public and private key pair and a series of ephemeral key pairs EKey = (keyId, ePubKey, ePrivKey) at *setup* stage where keyId is the ephemeral key identifier, ePubKey is the ephemeral public key, and ePrivKey is the ephemeral private key.

2) Obtain an ephemeral public key: For B to send a destructible message to A, it is necessary to obtain A's {keyId, ePubKey}, optionally signed with his long-term private key privKeyB, from A first.

3) Encrytpion: Using A's ephemeral public key ePubKey, along with B's long-term private key privKeyA, B can encrypt her message M with a secret session key S to obtain {M}S, calculate the HMAC value of M and S, and encrypt S and constraint C, optionally signed with privKeyA, with A's ePubKey to obtain {S, {C}privKeyA}ePubKey. Then B concatenates them with the corresponding ephemeral public key Id to form a single message encapsulation and sends it to A.

4) Decrytpion: After receiving B's encrypted message, the constraint interpreter inside A's mobile device can determine whether to decrypt {M}S by decrypting C and validating C's condition. If necessary, the message can even be deleted upon this stage.

B. *Enhanced Scheme*

1) Setup: As in the basic scheme. In addition, A sends {keyId, ePubKey}privKeyB to the Ephemeral Public Key Manager.

2) Obtain an ephemeral public key: For B to send a destructible message to A, it is necessary to obtain A's {keyId, ePubKey}privKeyB from the Ephemeral Public Key Manager first.

3) Encrytpion: As in the basic scheme.

4) Decrytpion: As in the basic scheme. In addition, A needs to upload a new pair of {keyId, ePubKey}privKeyB to the Ephemeral Public Key Manager after this stage.

C. Security Analysis

Note that message M is encrypted with sender's secret session key S, and S is encrypted with B's ephemeral public key ePubKey. Thus when B's ephemeral private key ePrivKey is deleted, M will become unrecoverable even if B's device is compromised. This is because no one can decrypt {S, {C}privKeyA}ePubKey to obtain S, with which to obtain M. If we use a long-term public key to protect S, e.g. {S}privKeyA, we cannot guarantee M is unrecoverable forever because an attacker can compromise B's device to get her long-term private key and decrypt {S}privKeyA to get S, with which to obtain M. Alternatively, instead of ePubKey, a session key can also be used to establish a secure two-party communication and ensure M's unrecoverability (by deleting M after it expires). However, when the receiver is off-line, we may need some special effort to upload the authenticated Diffie–Hellman key exchange pair to the key server to complete the session key negotiation process. We therefore, without loss of generality, elect the current scheme.

IV. DESTRUCTIBLE MESSAGE ENCAPSULATION FORMAT

In order to exchange our system-specific messages, we adopt JSON [11] to describe and encapsulate the encrypted sensitive data along with the public information, e.g. public key and key identifier, into a single message encapsulation, called the Instant Message as described below.

A. Public_Key_Request

Fig. 3 shows the content of a long-term public key request message sent to the receiver. The "InstantMessage" field is an identifier which indicates this message is for the Instant Messaging-based applications. The "version" field is reserved for further upgrade. The "type" field is for the purpose of this message.

B. Public_Key_Response

A sample of the long-term public key response message received from the receiver is shown in Fig. 4. The "version" and "type" fields are the same as in the request message. The "public_key" field is the content of the receiver's long-term public key, an OpenPGP [12] public key in this example. OpenPGP is used because it supports the passphrase protection for the OpenPGP private key which is required when we store the private key in receiver's mobile device.

C. Ephemeral_Public_Key_Request

Fig. 5 shows the content of an ephemeral public key request message sent to the receiver. The "version" and "type" fields are the same as in the first message.

D. Ephemeral_Public_Key_Response

A sample of the ephemeral public key response message received from the receiver is shown in Fig. 6. The "version" and "type" fields are the same as in the first message. The "ephemeral_public_key" field includes two sub-fields: "key_id" and "public_key" fields are the identifier and the content of the receiver's ephemeral public key, respectively.

session key and a time-based Instant Constraint created by a sender to claim that a message tied to this constraint should be deleted after 24 hours. Note that in a practical implementation, the sender adds the sum of the current NTP (Network Time Protocol) time plus 24 hours to the "time" field. Then the receiver should check the NTP time as well to avoid the situation that either the sender or the receiver doesn't set the correct time in their mobile devices.

V. IMPLEMENTATION AND EVALUATION

E. Ephemeral_Encryption_Data

A sample of the ephemeral encryption data received from the receiver is shown in Fig. 7. The "version" and "type" fields are the same as in the first message. The "ephemeral_encryption_data" field includes four sub-fields: "ciphertext" field is the sender's message encrypted with a secret session key, i.e. {M}S in Sec. III; "hmac" field is the HMAC value of M and S; "encrypted_secrets" field is a JSON object encrypted with the receiver's ephemeral public key, i.e. {S, C}ePubKey in Sec. III; and "ephemeral_ public_key_id" field is the identifier of the receiver's ephemeral public key. When the receiver's corresponding ephemeral private key is deleted, no one can decrypt the "encrypted_secrets" data to obtain the secret session key, and thus the ciphertext is unrecoverable. Note that we say the sender's message is destructed or deleted in this situation.


```
{"InstantMessage":

{
"version": 1,
"type": "Ephemeral_Encryption_Data", "ephemeral_encryption_data":
{
"ciphertext": "kfhfMHrkPWv2ANhs6JNlxvI1yxAHs1z0YvuqNfSUjTo=", "hmac": "+dwDkl9FXZUl+reslxNlSwHm/B4=",

"encrypted_secrets": "-----BEGIN PGP MESSAGE-----\nVersion: BCPG v1.46\n\nhIwDpk7XyicVRCMBA/9e9DACSCwQ/GVA5kzXBZFWRenjBubgTNnhYDlW
RotOwyId\nj7Ar…(skip)…Vf+q7HKUg==\n=MDhg\n-----END PGP MESSAGE-----\n",
"ephemeral_public_key_id": "e86f6960b54e48279efa359643daa135"
}
}
```


Figure 7. Message of ephemeral encryption data

Regarding when to delete the ephemeral private key, we define the "constraint" field as a JSON object and the sender

The Instant Messaging uses the customized messages (i.e. the Instant Message encapsulation format) atop any existing instant messaging services to make the transmitted messages secure and self-destructible. Thus, there's no special-purpose dedicated server needed. We describe our Android implementation using the basic scheme in Section III and give the performance evaluation as follows.

A. Implementation

To demonstrate the feasibility of our proposed Instant Messaging architecture, we have developed a mobile application on an Android 2.3 mobile phone. The screenshots are shown in Fig. 9.

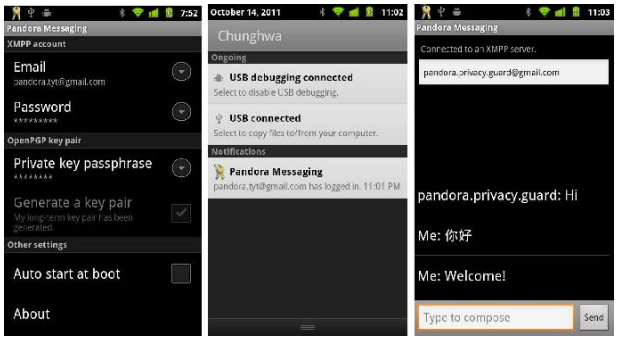

Figure 9. The screenshots of the Instant Messaging based Android app

For the instant messaging communication, we adopt XMPP instant messaging open standard [9] and use the XMPP-compliant Google Talk service as the instant messaging server. All Gmail accounts are also the XMPP accounts which can login to the Google Talk service. For the XMPP client API on Android, we adopt an Android version of Smack [13], an open source XMPP client library, for our implementation.

Note that our system architecture is mainly for protecting the XMPP client's messages from being recovered by any unrelated parties, or even the XMPP server. To make the messages more secure, we can further adopt Transport Layer Security (TLS) [14] and Simple Authentication and Security Layer (SASL) [15] atop the XMPP network for client to server encryption and authentication.

Regarding the encryptions, we adopt AES (Advanced Encryption Standard) 128-bit key's symmetric encryption (AES/CBC/PKCS5Padding) for the message encryption [16], and adopt HMAC (Hash-based Message Authentication Code) with SHA-1 for verifying both the data integrity and the authenticity of a message [17]. Besides, we adopt OpenPGP with RSA 1024-bit asymmetric encryption algorithm to encrypt the AES key and the message constraint (see Sec. III for more information). Note that the OpenPGP keys are ephemeral keys in our Instant Messaging architecture. We adopt the Bouncy Castle Java cryptography APIs [18] to support the above cryptographic techniques in our implementation.

B. Evaluation

To evaluate the performance of our proposed Instant Messaging architecture, we deploy our mobile application on an Android phone equipped with a 1 GHz Qualcomm processor. Fig. 10 shows the average time of generating the OpenPGP key pairs using RSA encryption algorithm for 50 runs. Note that the key size of 1024 bit is recommended by the RSA Laboratories for corporate use [19]. Although the 1024-bit OpenPGP key's average generation time, say 1011 milliseconds (ms), is not very short, it still has room for improvement if we use C/C++ to generate the keys instead of using Java in the current implementation. Besides, the ephemeral OpenPGP key is generated in the background process, so the execution time of one second is also acceptable. In our Instant Messaging architecture, we only generate a new ephemeral OpenPGP key after the previously generated ephemeral OpenPGP key is requested and used by the sender. Thus, it is safe to say that the key generation does not tax the system too much.

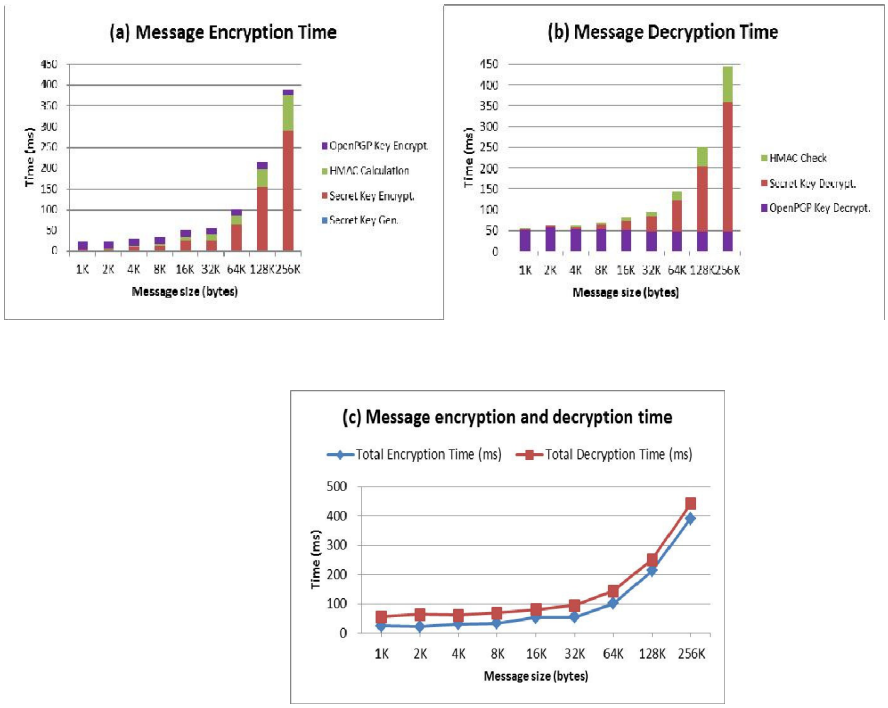

Figure 10. The average time (in milliseconds) of generating the OpenPGP key pairs using RSA algorithm for 50 runs

Fig. 11 (a) shows the detailed encryption time, including time for generating a 128-bit AES secret key, using the secret key to encrypt different size of messages, calculating the HMAC value of the secret key and the message, and using a 1024-bit OpenPGP public key to encrypt the message constraint and the secret key. Fig. 11 (b) shows the corresponding detailed decryption time excluding the secret key generation time. The average time for generating a secret key is 0.54 ms, so it is negligible. The average time for encrypting a message with a secret key increases as the message size grows. The average HMAC calculation time also increases as the message size. The OpenPGP key encryption/decryption time is a fixed value because the encrypted/decrypted target is the constraint and the secret session key. And the OpenPGP key decryption time (average 50.40 ms) is about three times of the encryption time (average 16.84 ms).

Fig. 11 (c) shows the comparison of the message's total encryption and decryption time. The average encryption

or decryption time of small messages of size smaller than 32 Kbytes is less than 100 ms, so that users would not feel the performance penalty.

From the experimental results, we conclude that our proposed Instant Messaging architecture doesn't degrade the performance significantly in terms of the message encryption and decryption time. The only overhead is the ephemeral key generation time, which can be further reduced as mentioned earlier.

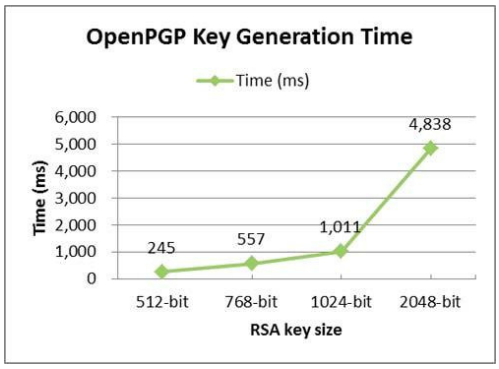

Figure 11. The average time (in milliseconds) of the message (size in bytes) encryption and decryption for 50 runs

VI. Related Work

Perlman [5][6] presented a scheme in which a service known as an Ephemerizer creates encryption and decryption functions with expiration time. If the clients, when creating and reading messages, use special software that does not store decrypted messages in a stable storage, expired messages will be unreadable once the Ephemerizer deletes expired keys.

Vanish [7] is a proof-of-concept prototype system based on the P2P distributed hash tables (DHTs). Vanish inherits some concepts from the aforementioned Ephemerizer paper, but the responsibility of key management is transferred from Ephemerizer(s), which is a dedicated server or a group of servers, into the P2P network. The Vanish causes sensitive information to irreversibly self-destruct, without any action on the user's part and without any centralized or trusted system. Nevertheless Unvanish [20] has proven that the initial Vanish implementation cannot withstand the Sybil attacks. Besides, it is almost impossible to precisely determine the time period that a threshold subset of P2P nodes stay in the network. Thus, there is no precise guarantee on the expiration time of the sensitive data according to [21].

With regard to the usage of instant messaging, both of Apple's MobileMe [22] and Google's Android C2DM [23] services are using the instant messaging protocols, AIM (OSCAR) protocol and open-standard Jabber (XMPP) protocol respectively, to support the Cloud synchronization and mobile push behaviors. Therefore we can use instant messaging not only to send the text messages but also to support the application-level data synchronization on mobile devices.

VII. Conclusion

The user using the existing instant messaging services is subjected to constant eavesdropping, unwanted leakage of confidential information. Besides, they cannot control the lifespan of their outgoing messages. To solve these issues, we introduce the self-message-destructing feature atop the instant messaging services. Self-destructible messages stored in a mobile device can be considered to satisfy the expiration condition instantly, so the messages become unrecoverable or useless. This 'disabling' reuse is particularly important when one wants to protect personal information.

We propose the Instant Messaging, an enhanced secure instant messaging architecture which is equipped with a self-message-destructing feature for sensitive personal information applications in a mobile environment. The Instant Messaging-based system enables senders to set time, frequency, and location constraints. These conditions determine when the transmitted messages should be destructed and thus become unreadable for receivers. The Instant Messaging-based system securely sends self-destructible messages to receivers in a way that it uses ephemeral keys to encrypt the messages and transmits the encrypted messages to the designated receiver via the XMPP instant messaging service in real time. When the transmitted messages' constraints are satisfied, the ephemeral key used for encryption will be deleted, and thus the encrypted messages become unrecoverable. We have implemented a simple messenger application on the Android platform and have evaluated its performance to show that our proposed Instant Messaging architecture is practical and feasible for sensitive personal information communication on mobile devices.

REFERENCES

[1] M. Mannan and P. C. Van Oorschot, "Secure Public Instant Messaging: A Survey", in Privacy, Security and Trust, 2004.

[2] Declan McCullagh, "How safe is instant messaging? A security and privacy survey", CNET News website. http://news.cnet.com/8301-13578_3-9962106-38.html

[3] H. Kikuchi, M. Tada, and S. Nakanishi, "Secure instant messaging protocol preserving confidentiality against administrator," in 18th International Conference on Advanced Information Networking and Applications, 2004 (AINA 2004), vol. 2, pp. 27- 30.

[4] D. Boneh and R. J. Lipton, "A revocable backup system," in Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6, San Jose, California, 1996, pp. 91–96.

[5] R. Perlman, "The Ephemerizer: Making Data Disappear", Journal of Information System Security 1(1), 51–68 (2005)

[6] R. Perlman, "File system design with assured delete", In SISW 2005 Proceedings of the Third IEEE International Security in Storage Workshop, pp. 83–88. IEEE Computer Society, Los Alamitos (2005)

[7] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy, "Vanish: Increasing Data Privacy with Self-Destructing Data", In Proc. of the 18th USENIX Security Symposium, 2009.

[8] C. Pöpper, D. Basin, S. _apkun, and C. Cremers, "Keeping data secret under full compromise using porter devices," in Proceedings of the 26th Annual Computer Security Applications Conference, Austin, Texas, 2010, pp. 241–250.

[9] XMPP Standards Foundation. http://xmpp.org/

[10] Google Projects for Android. http://code.google.com/android/

[11] JSON web site. http://www.json.org/

[12] The OpenPGP Alliance. http://www.openpgp.org/

[13] Smack, an open source XMPP (Jabber) client library. http://www.igniterealtime.org/projects/smack/index.jsp

[14] A. Melnikov and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006. http://tools.ietf.org/html/rfc4422

[15] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, Auguest 2008. http://tools.ietf.org/html/rfc5246

[16] FIPS PUB 197, NIST, "Advanced Encryption Standard (AES)", November 26, 2001.

[17] H. Krawczyk , M. Bellare , and R. Canetti , "RFC2104: HMAC: Keyed-Hashing for Message Authentication", February 1997.

[18] Bouncy Castle Java cryptography APIs. http://www.bouncycastle.org/java.html

[19] RSA Laboratories, "How large a key should be used in the RSA cryptosystem?" http://www.rsa.com/rsalabs/node.asp?id=2218

[20] S. Wolchok, O. S. Hofmann, E. W. Felten, J. A. Halderman, C. J.Rossbach, B. Waters, and E. Witchel. "Defeating Vanish with low-cost Sybil attacks against large DHTs", In Proc. of NDSS, 2010.

[21] Qiang Tang, "From Ephemerizer to Timed-Ephemerizer: Achieve Assured Lifecycle Enforcement for Sensitive Data". http://doc.utwente.nl/69264/

[22] Apple's MobileMe website. http://www.me.com/

[23] Google's Android Cloud to Device Messaging Framework. http://code.google.com/e/android/c2dm/