# Reducing User Error by Establishing Encryption Patterns

Anthony Gabrielson, Haim Levkowitz
Department of Computer Science
University of Massachusetts Lowell
One University Avenue, Lowell, MA 01854, United States of America
agabriel@cs.uml.edu
haim@cs.uml.edu

*Abstract*— **This paper is motivated by a desire to create a user friendly, technically flexible approach to cryptography. This approach can create a flexible design pattern that is useable under a wide variety of circumstances from broadcast media to existing network protocols. This is accomplished by designing a new, key piece of infrastructure used for cryptography key lookup. Once the key is obtained the application/protocol developer is given the flexibility to meet their requirements.**

***Keywords-Security Patterns; Patterns of Trust; Authorization Patterns***

## I. INTRODUCTION

Encryption technology needs to evolve to a point where end-users are not aware they are using it [1]. To accomplish this goal a common infrastructure, which can be used across a wide spectrum of applications, is needed. Currently deployed technologies, namely Secure Sockets Layer (SSL) and Certificate Authorities, are no longer sufficient because in a quest for the lowest possible price the current infrastructure has shown it is extremely vulnerable to attacks due to woefully inadequate administrative procedures [2]-[3] and technical issues that prevent it from being used to solve an array of problems from email to media distribution.

This paper presents a solution that is extremely configurable and distributable. It develops a security pattern by leveraging existing technologies and ideas, and using them in a manner similar to that of other widely deployed technologies that make up the fundamental building blocks of the Internet. The structure of this solution also allows developers to re-examine other features in the TCP/IP suite, like UDP broadcast and multicast, enabling easier distribution of encrypted content to many users simultaneously.

The rest of the paper is organized as follows. The remainder of the introduction defines trust and requirements. The foundations for the general implementation of our pattern is described in Section II. Section III describes our concept of operations for a TCP and UDP implementation. The implementation and testing details for our pattern are described in Section IV. Sections V-VI describe the benefits and shortcomings of this pattern. Future improvements are described in Section VII. The conclusion follows in Section VIII.

### A. Trust

The main goal of this research has been to establish a trusted encrypted channel between two points that is easily configurable by the developer while requiring minimal user input. The channel must not work if there is any breakdown in trust. To accomplish this we first need to define trust. A proper trust definition is essential to understanding requirements. This is the definition of trust we have been working with:

> *"Trust is a mental state comprising: expectancy — the [trusting individual] expects (hopes for) a specific behavior from the [entity s/he is putting trust in] (such as providing valid information or effectively performing cooperative actions); belief — the [trusting individual] believes that the expected behavior occurs, based on the evidence of the [trusted entity's competence] and goodwill; and willingness to take risk — [based on that belief] the [trusting individual] is willing to take risk [for the purpose of achieving some desired result.]"* [4]

### B. Requirements

Our requirements are fairly straightforward. The proposed solution only requires a minimal amount of setup information to establish an encrypted channel. If there are any key or encrypted channel problems the recipient shall not be able to decrypt the data. The encryption algorithm needs to support encryption with multiple public keys. The implementation also needs to be able to support non-malleability [5], non-repudiation, and authentication.

The added interaction between the user and the client application required for cryptography needs to be minimal. We focus on an API that is used at the application layer of the ISO network model; our approach utilizes a separate key server, and requires a small amount of additional information, almost all of which can be automated, to the client application: 1. the location where the key server resides on the network; 2. a local username; and 3. the recipient's username and host. The key server details can be hidden by creating an additional user variable on the local system or creating a new DNS record type that specifically points to the key server. The local username is also typically available as a user variable. The recipient's username could also be standardized under certain circumstances with a *getservbyname()* [6] system call. This could reduce the

 ISBN: 978-1-61208-158-8 

required user input to just one piece of information -- the destination host name, which is already required.

Cryptography software developers have largely ignored the TCP/IP broadcast and multicast features; however this important functionality needs to be considered [7]. To help facilitate those needs, copyright holders increasingly need to be able to easily distribute intellectual property while ensuring it is protected. Software should be able to encrypt the same data with multiple public keys and transmit it such that only the intended recipients will be able to decrypt it. This capability, combined with an understanding of the network media reliability allows broadcast and multicast features to be utilized with cryptography.

Our proposed solution also needs to support non-repudiation and authentication; of these two, the more important one is non-repudiation: It is extremely important that data sources cannot be disavowed. While key-based authentication is desirable it may be appropriate in some situations to add an additional level of trust; an application developer may wish to also add a username and a password or a secondary key file, such as a crafted image, that will only be accepted from a designated user.

## II. PROPOSED SOLUTION

### A. General Encryption Technology

Currently, only the PGP/GPG cryptography standard meets the encryption requirements of this project. However, it does have a few trade-offs that must be handled. One major trade-off is the zip compression algorithm utilized by the PGP standard, as illustrated in Figure 1. Since PGP uses compression, a particular data size to cipher text size cannot be guaranteed. We address this later in this paper.

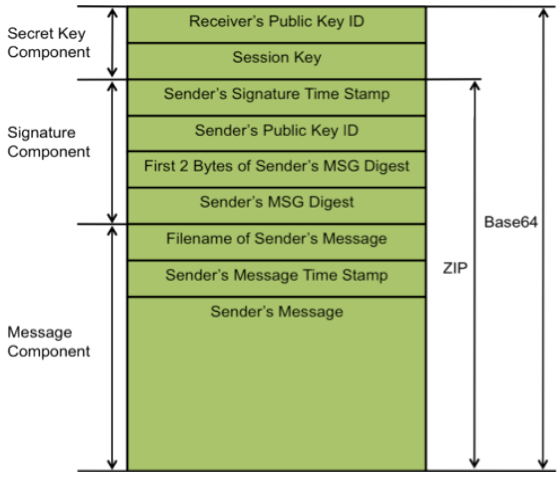


Figure 1: PGP Data Shape [8]

### B. General Terms

There are three terms that will appear throughout the remainder of this paper – *Originator, Destination*, and *Encryption Key System (EKS).* All data and connections originate from an Originator, which could be a Client or a Server depending on protocol configuration. The Encryption Key System (EKS) serves public cryptography keys (Pk) to the Originator. It works similar to the Domain Name System (DNS) [9] except that rather than providing IP addresses it will provide public keys.

### C. Web of Trust Model

Lucas tells us that PGP goes out of its way not to define trust [10], which leads to an application that is difficult to use properly [1]. A trust (in performance and belief) model needs to be established that automatically finds and trusts keys. The trust in performance is based on the destination's ability to pass a challenge by decrypting the received data. The trust in belief is in the process of receiving the correct public key in the process. If both a trust in performance and belief are satisfied then public keys should be fully trusted.

Huang has defined a semantic for a belief relationship [4]:

$$\tau\beta(\delta,\varepsilon,\xi,\kappa) \equiv \alpha(\varepsilon, \kappa \rightarrow \xi) \supset \psi(\delta,\kappa \rightarrow \xi) \qquad (1)$$

where a trust-in-belief relationship, $\tau\beta(\delta,\varepsilon,\xi,\kappa)$, represents that trusting-entity $\delta$ trusts trusted-entity $\varepsilon$ regarding $\varepsilon$*'s* belief $\xi$ in context $\kappa$. This trust relationship means that if $\varepsilon$ believes $\xi$ in context $\kappa$, then $\delta$ also believes $\xi$ in that context [4].

A semantic for a performance relationship can be formulated as [4]:

$$\tau\Pi(\delta,\varepsilon,\xi,\kappa) \equiv \alpha(\varepsilon, \kappa \rightarrow \xi) \supset \psi(\delta,\kappa \rightarrow \xi) \qquad (2)$$

where a trust-in-performance relationship, $\tau\Pi(\delta,\varepsilon,\xi,\kappa)$, represents that trusting-entity $\delta$ trusts trusted-entity $\varepsilon$ regarding $\varepsilon$*'s* performance $\xi$ in context $\kappa$. This relationship means that if $\varepsilon$ makes $\xi$ in context $\kappa$, then $\delta$ believes $\xi$ in that context [4].

Trust can now be directly defined such that:

$$\tau(\delta,\varepsilon,\xi,\kappa) \equiv \tau B(\delta,\varepsilon,\xi,\kappa) \cap \tau\Pi(\delta,\varepsilon,\xi,\kappa) \qquad (3)$$

That is to say that if the destination believes it is communicating with the correct originator and that originator is performing as expected it would be trusted.

### D. Web of Trust in Practice

In practice, the trust relationship begins with the local EKS Server. Figure 2Figure 2 illustrates the communication that might be required for an originator to obtain a particular key that is not hosted on a Local EKS server. If the Pk were hosted locally the Local EKS Server would simply return it. However in this case the originator will connect to the Local EKS Server that will locate and provide the Originator with the desired Pk.

 ISBN: 978-1-61208-158-8 

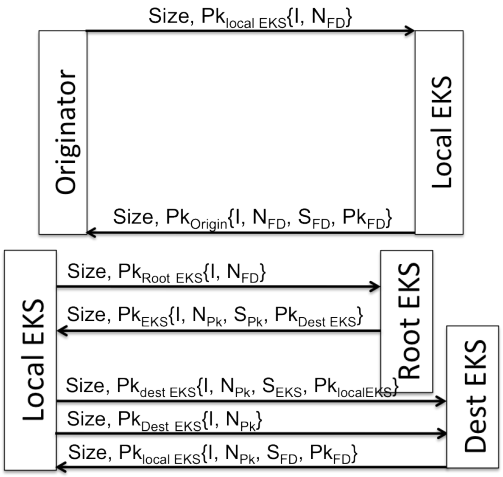


Figure 2: Web of Trust

The Local EKS Server starts this process by communicating with the appropriate Root EKS Server that will provide the Local EKS Server with $Pk_{Dest\ EKS}$. The Local EKS Server will then contact the Destination EKS Server asking for $Pk_{FD}$ or Final Destination; the Final Destination is not illustrated in the figure. Since the Destination EKS Server does not have $Pk_{local\ EKS}$ it is sent during the initial connection. The process of looking up a Pk could best be compared with looking up an IP address. Once the IP address has been obtained the originator can begin to establish communication with the destination host.

The Originator trusts the key provided by the Local EKS server because of the process required to retrieve it. The Originator and Local EKS Server have a priori knowledge of their Pks. The Local EKS Server has the same a priori knowledge with the Root EKS Server. The Local EKS Server only needs to send its Pk to the Destination EKS Server. This process has three main strengths 1. it only exposes the ability to encrypt a message that can only be decrypted by the destination; 2. IP addresses can still change since key lookup is based on name; 3. if a Destination's IP address is spoofed the Local EKS Server will not have access to the spoofed Destination's Pk at lookup; so the spoofed Destination will be unable to decrypt any received data. The Originator trusts the received Pk because those three characteristics create a trust in belief and performance.

## III. CONCEPT OF OPERATION

### A. TCP Example

Let us walk through the implementation of an FTP like protocol using TCP. The Originator, in this case the Client, sends a request to the EKS Server asking for the Destination, or Server's, Pk. The EKS Server responds with the Destination's key name ($N_{Pk}$) and it's $Pk_{Server}$ – all encrypted with the $Pk_{client}$. The Originator is now ready to initiate encrypted communication with the Destination. The Client opens a connection and sends an encrypted IDENT message type that includes the $Pk_{client}$. The Destination then decrypts and installs $Pk_{client}$ and establishes trust in it. A two-way encrypted communication channel between the Client and the Server is now established. The Client follows up on its initial connection with a request in field I, in the second communication between the client and server; since the Server has installed and trusts $Pk_{client}$, which is the first communication between the client and server, it is able to respond as illustrated in Figure 3***Error! Reference source not found.***.

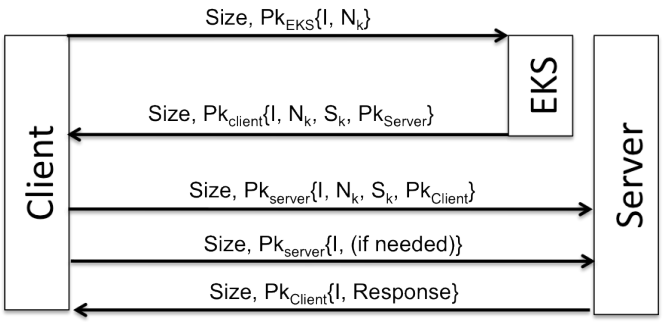


Figure 3: TCP Example

### B. UDP Example

Let us now walk through a protocol that implements the UDP broadcast capability. The Originator or Server in this case, sends a request to the EKS Server asking for the Destination, or Client's, Pk. The EKS Server responds with the Destination's $N_{Pk}$ and $Pk_{client}$ – all encrypted with $Pk_{server}$. The Originator installs $Pk_{client}$ and is now ready to establish an encrypted one-way channel with the Destination as illustrated in Figure 4**Error! Reference source not found.**. Since the Client will never need to reply in this scenario, the Server does not provide $Pk_{server}$ to the Client.

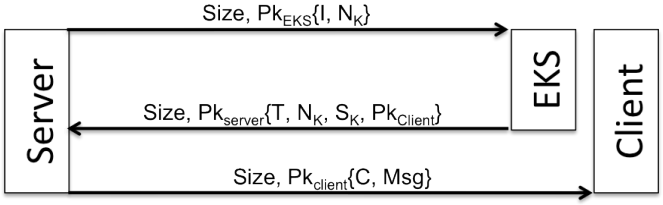


Figure 4: UDP Example

To work within UDP limitations it is recommended that data be parsed such that its encrypted size is less then or equal to the size of the Maximum Transmission Unit (MTU) [11] of the particular network medium. A counter has also been inserted to enhance the resolution of the PGP time stamp illustrated in Figure 1Figure 1 to better enable any data buffering implemented at the destination.

## IV. IMPLEMENTATION AND TESTING

### A. Initial Setup

A few steps are necessary to set up the infrastructure required by the EKS Server, Originator, and the Destination. While these steps may seem in depth this process can and

should be automated for a final solution. First, all three accounts must initialize PGP keys.

The EKS Server must have both $Pk_{Destination}$ and $Pk_{Originator}$ installed and trusted in its key ring. The Originator must initially have and trust $Pk_{EKS\ Server}$. The EKS Server will provide $Pk_{Destination}$ upon request. After the EKS Server transfers $Pk_{Destination}$ the Originator will install and trust it. The Destination does not initially need any Pks installed in its key ring because it will not initiate contact with any other host. When the Originator initially connects $Pk_{Originator}$ will be transferred during the initial message if needed. The Destination will trust and install $Pk_{Originator}$ if it is transferred.

### B. Implementation

We have implemented this solution architecture using both the 32- and 64-bit versions of the Ubuntu Linux 10.10 operating system hosted on VMware Workstation and Fusion; we see no reason this solution will not work on other platforms like Microsoft Windows or Apple's Macintosh OS X. GPG version 1.4.10 was used and installed using the apt tool. We used Qt 4.7, a cross-platform application framework, to construct these two tools together. We implemented the TCP and UDP communication protocols. This solution makes communication with a spoofed IP address impossible because an attacker will not have access to the required secret keys (Sk) $Sk_{Originator}$ or $Sk_{Destination}$ to decrypt the transmitted data.

### C. Testing

There can be many use cases for this research; we tested two of them: TCP, and UDP Broadcast Communication. In both cases all three hosts were on the same local network. The EKS Server trusted $Pk_{EKS}$, $Pk_{Destination}$, and $Pk_{Originator}$. The Originator trusted two public keys before running – its own and $Pk_{EKS}$; after connecting to the EKS Server the Originator also trusted $Pk_{Destination}$. The Destination initially trusted one key – its own; after the Originator connected to the Destination it trusted $Pk_{Originator}$ if transmitted.

### D. GPG Commands

A small number of `GPG` commands were needed for this implementation [12] and we will highlight a few of the more compelling features. The `encrypt` capability includes the –s and –r options. The –s option signs the data with the sender's key, which is helpful for TCP communication; assuming the Destination(s) are familiar with the Originator, this can be used for sender verification since the recipient will understand the sender's signature. The –r option allows users to be added to the encryption stream and can be called multiple times with different usernames enabling a UDP broadcast capability where only the intended recipients will be able to decrypt the stream.

## V. BENEFITS

The implementation described provides several key benefits to existing solutions. It enables non-repudiation and authentication through the built-in PGP capabilities and a defined trust relationship. The user provides minimal input to start and maintain an encrypted channel, thus simplifying the end-user's experience [1] by providing fewer opportunities to make a mistake. Our solution is more scalable than other key distribution techniques as explained below. Finally, communication is task-based, allowing better protection of information, such as usernames and/or passwords, because the API is used at the Application Layer.

Since PGP enables non-repudiation and authentication and we have established an automated trust definition users can connect to a host and trust that they have in fact connected to the right host because the EKS Servers are trusted and match DNS results. Our defined trust relationship may in some cases delude the trust PGP originally intended so usernames and passwords can be used additionally and linked to a particular key signature, increasing the overall security.

The user is never asked any security questions, unless the implementation utilizes the added security of a username/password. If a particular host is unable to communicate it may indicate that an attack is underway and communication at this point in time is not desirable. In this case a malicious host will not be able to decrypt the data since it does not posses the correct private key required to decrypt the data. The method to download new public keys establishes trust and there is no way to compromise data integrity. This feature prevents attack vectors like evilgrade [13], presented by Amato, since there is no way to decrypt the transmitted network traffic.

The EKS server and protocol design have several benefits when compared to a Kerberos Key Distribution Center [14] implementation. The two primary differences are: first, EKS servers can be scaled up to cache keys and thus reduce the expense of continuously looking up the same keys, similar to DNS capabilities. The other primary difference is that the EKS server does not need to contact both the Originator and Destination with a ticket since the Originator will send the Destination its key during the initial connection if needed. The added capabilities of this solution allow it to be more scalable than the alternative.

Protocols can become more task-oriented, enabling better sandboxing [15]. Since each task or daemon can use its own private key, tasks can be better partitioned on the same system. Information, like usernames and/or passwords, can be encrypted for transit so that it cannot be sniffed [16], which will eliminate the Man-in-the-Middle Attack [17]. Data can be stored in the cloud in a manner that the service provider, or any attacker, will not have access to it.

## VI. SHORTCOMINGS

Our proof-of-concept exhibits two main shortcomings. All Originator traffic must start with its home EKS Server. The current implementation is performance-limited due to file IO; GPG requires that all encryption and decryption activity go through a file.

All Originator communication must start with its EKS Server because of the trust relationship between the two. If users are mobile and using random access points to access the Internet their home EKS Server must be publicly available on the Internet. This only poses a minimal security

 ISBN: 978-1-61208-158-8 

risk since the EKS Server is a low value target since it only hosts and transmits public keys.

The current GPG implementation is extremely file IO dependent; all communications are realized through a file on both sides of the connection. This can have an impact on the speed of the communications. This was observed during early testing on a Windows XP system, where a noticeable delay was perceptible. The remainder of the implementation and testing was done on a Linux system in a virtual machine and file IO was not a perceptible slow down, however simultaneous connections were limited. We currently view this limitation as minimal since it could easily be resolved with a GPG API.

## VII. FUTURE IMPROVEMENTS

This proof-of-concept still needs improvements. First and foremost our research has so far focused on two use cases and more are needed to address other scenarios. In addition, there are several other areas that also need improvement, such as, key caching time limits, and a GPG API to avoid file thrashing.

More use cases need to be added to the EKS Server. There are two main EKS Server use cases -- when the Originator and Destination are on the same network and when they are further apart on the Internet. Additional protocols need to be implemented for the second use case since the local EKS Server will need to be able to communicate with other EKS Servers to look up other public keys.

After more use cases are added the EKS Server needs a time based cache table added for non-local hosts. Similar to DNS [9], an EKS Server should cache non-local Pk for a certain amount of time to reduce network traffic and CPU utilization. This feature is also a security enhancement since key pairs could be allowed to sunset after a particular period of time.

This implementation could also be considerably quicker with GPG provided API level access; this would eliminate almost all file IO transactions. A linkable library that provides the same capability but provides byte level access to data input and output would accomplish this.

## VIII. SUMMARY

The solution proposed in this paper forms the foundation of a new cryptography design pattern. This design pattern allows developers the flexibility they require and users the security and simplicity they need to accomplish a task. This pattern is only possible because of a proper trust definition that enables security issues to be handled automatically.

Modern computing is already complex enough without adding more layers that users need to worry about. Application developers need secure flexible cryptography libraries that implement best practices that are easy to use to solve encryption problems from media distribution to email. Encryption technology needs to become ubiquitous in computing. To create this capability the solution needs to be as close to invisible as possible to the end user; the user must be able to focus on the task at hand. While the user is focusing on her/his task any security-related questions necessary to achieve these goals must be automatable to avoid overwhelming the user.

To become more ubiquitous the solution needs to be flexible enough to enable as many existing data distribution capabilities as possible while restricting decryption privileges to only the intended recipients. Once this goal is achieved many of the existing problems currently being experienced will be obsolete.

## REFERENCES


[1] A. Whitten. (1999). Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In SSYM'99 Proceedings of the 8th conference on USENIX Security Symposium, ACM, Ed. Vol. 8. 15.

[2] E. Nakashima. (2011) Washington Post webpage on Cyberattack. [Online]. Retrieved June 17, 2011, From The Washington Post: http://www.washingtonpost.com/world/us_agencies_respond_to_cyberattack_on_information_security_firm/2011/03/23/ABDhjoKB_story.html?wprss=rss_homepage

[3] F. Rashid. (2011) eWeek webpage on Fake SSL certificates, [Online]. Retrieved June 17, 2011, From eWeek.com: http://www.eweek.com/c/a/Security/Fake-SSL-Certificate-Incident-Highlights-Flaws-in-DNS-Comodo-CEO-440985/

[4] J. Huang. (2010). A formal-semantics-based calculus of trust. In Internet Computing, IEEE. iEEE, 38 – 46.

[5] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. (1998). Relations Among Notions of Security for Public-Key Encryption Schemes. In Advances in Cryptology – Crpto '98.

[6] FreeBSD Man Page., [Online]. Retrieved June 17, 2011, From The UNIX.com Man Pages: http://www.unix.com/man-page/FreeBSD/3/getservbyname/

[7] Lixin Gao; Towsley, D. ; , "Threshold-based multicast for continuous media delivery," *Multimedia, IEEE Transactions on* , vol. 3, no. 4, pp. 405-414, Dec 2001

[8] J. Wang. (2009). Computer Network Security Theory and Practice. Springer.

[9] Albitz, Paul. 2001. DNS and BIND, 4th Edition. Oreilly.

[10] M. Lucas. (2006). PGP & GPG Email for the practical paranoid. No Starch Press.

[11] Reviriego, P. ; Sanchez-Macian, A. ; Maestro, J.A. ; Bleakley, C.J. ; , "Increasing the MTU size for Energy Efficiency in Ethernet," *Signals and Systems Conference (ISSC 2010), IET Irish* , vol., no., pp. 124-128, 23-24 June 2010

[12] gnupg: Documentation, [Online]. Retrieved June 17, 2011, From The GNU Privacy Guard: http://www.gnupg.org/documentation/manuals/gnupg-devel/GPG-Configuration-Options.html#GPG-Configuration-Options.

[13] F. Amato. Evilgrade: you have pending upgrades... . Ekoparty Security Conference, Buenos Aires, Argentina, Nov 30 – Dec 1, 2007.

[14] Neuman, B.C.; Ts'o, T.; , "Kerberos: an authentication service for computer networks," *Communications Magazine, IEEE* , vol. 32, no. 9, pp. 33-38, Sep 1994.

[15] Greamo, C.; Ghosh, A.; , "Sandboxing and Virtualization: Modern Tools for Combating Malware," *Security & Privacy, IEEE* , vol.9, no. 2, pp. 79-82, March-April 2011.

[16] Qadeer, M.A.; Zahid, M.; Iqbal, A.; Siddiqui, M.R.; , "Network Traffic Analysis and Intrusion Detection Using Packet Sniffer," *Communication Software and Networks, 2010. ICCSN '10. Second International Conference on* , vol., no., pp. 313-317, 26-28 Feb. 2010

[17] M. Marlinspike. (2009). Defeating SSL. In Black Hat DC.


 ISBN: 978-1-61208-158-8 