

TALUG Meeting Notes

November 23, 2007

PGP and Cryptography: Presented by Loren Weith

Loren Weith works as a Senior Network Engineer for Centracomm Communications in Findlay, Ohio and is currently finishing up his masters degree in Computer Science with a specialization in Information Security Technology from Eindhoven University of Technology in The Netherlands.

Background Explanation

In order to explain PGP, there is some background information that needs to be understood. In general, there are two types of cryptography: *Symmetric* and *Asymmetric*.

Symmetric Encryption

Symmetric encryption uses a pre-shared key method, where the same key is held by both the sender and receiver. This key can both encrypt and decrypt the message. In order for this method to work, the key would have to be securely exchanged, possibly by a physical meeting. If at any time the shared key becomes public (or at least not secret) the encryption is compromised. Symmetric encryption can be summarized as:

- Computationally efficient
- Same key encrypts and decrypts
- Absolute secrecy of the key is required
- Keys must be distributed before encryption can be used

How It Works

E = Encryption Function

D = Decryption Function

K = Secret Key

P = Plaintext Message

C = Encrypted Message

$$E(P, K) \rightarrow C$$

$$D(C, K) \rightarrow P$$

Asymmetric Encryption

Asymmetric encryption (also known as public key cryptography) uses key pairs. Each user has both a private key and a public key, which are mathematically related. The public key is publicly distributed, and the private key is kept secret. Unlike symmetric encryption, no private key exchange is required.

- Computationally expensive
- Different keys encrypt and decrypt
- Public keys can be publicly disclosed
- Private keys are never transferred

How It Works

P = Public Key

S = Secret (Private) Key

$$P(S(x)) = x$$

$$S(P(x)) = x$$

Note: In terms of cracking the encryption, 128 bit symmetric encryption is approximately as strong as 1024 bit asymmetric encryption.

PGP Encryption

Using PGP Encryption

By using a combination of symmetric and asymmetric encryption, we can take advantage of the benefits of both. The premise is that we have two people, Alice and Bob. Alice wants to send an encrypted message to Bob. Both Alice and Bob have private and public keys.

In order to encrypt, Alice generates a random session key... the more random the better. The random session key is then used to symmetrically encrypt the message. The symmetric encryption is computationally inexpensive. Alice then encrypts the random session key with Bob's public key. This is asymmetric encryption, and is computationally expensive.

Once Bob receives his message, he asymmetrically decrypts the random session key with his private key, and then symmetrically decrypts the message with the random session key.

How It Works

E = Encryption Function

D = Decryption Function

P_a = Alice's Public Key

S_a = Alice's Secret Key

P_b = Bob's Public Key

S_b = Bob's Secret Key

$RSK_{Decrypted}$ = Random Session Key, Decrypted (Unencrypted)

$RSK_{Encrypted}$ = Random Session Key, Encrypted

$M_{Decrypted}$ = Message, Decrypted (Unencrypted)

$M_{Encrypted}$ = Message, Encrypted

Alice encrypts the message, and the random session key:

$$E(M_{Decrypted}, RSK_{Decrypted}) \rightarrow M_{Encrypted}$$

$$P_b(RSK_{Decrypted}) \rightarrow RSK_{Encrypted}$$

And emails both to Bob. Bob then decrypts the session key, and uses that to decrypt the message

$$S_b(RSK_{Encrypted}) \rightarrow RSK_{Decrypted}$$

$$D(M_{Encrypted}, RSK_{Decrypted}) \rightarrow M_{Decrypted}$$

Problems

- Is P_b really Bob's key?
- Is the $M_{Encrypted}$ really from Alice?

Digital Signatures and Hashes

A hash algorithm is a fingerprint for the message. It outputs a sequence of digits based on the message and a mathematical formula. Think of it as a "Hash Fairy" (laughter overheard) that returns a hash for a given input. If the messages are identical, the hash will be identical.

In a hash algorithm, we want the following properties:

- Collision Resistance – Identical hashes cannot easily be made from different messages. This is very broken for MD5.
- Preimage Resistance – Output of algorithm cannot easily be used to figure out what the input to the algorithm was. Intact (probably) for MD5.

Hashing example: Alice is sending the contents of *War and Peace* to Bob.

M = Message: content of *War and Peace*

$D_{Decrypted,Alice}$ = Message digest (hash), decrypted (unencrypted) as produced by Alice

$D_{Encrypted,Alice}$ = Message digest (hash), encrypted as produced by Alice

$D_{Decrypted,Bob}$ = Message digest (hash), decrypted (unencrypted) as produced by Bob

Alice computes the hash of her message, and then encrypts it with her secret key.

$$\begin{aligned} Hash(M) &\rightarrow D_{Decrypted,Alice} \\ S_a(D_{Decrypted,Alice}) &\rightarrow D_{Encrypted,Alice} \end{aligned}$$

Bob receives the message, and the encrypted hash. In order to verify the message has not been changed, and that it is from Alice, he computes the hash of the message. Bob then decrypts Alice's encrypted hash, and compares it to the hash he computed. If all is well, these two hashes will match.

$$\begin{aligned} Hash(M) &\rightarrow D_{Decrypted,Bob} \\ P_a(D_{Encrypted,Alice}) &\rightarrow D_{Decrypted,Alice} \\ D_{Decrypted,Bob} &= D_{Decrypted,Alice} \end{aligned}$$

Key Strength

PGP Encryption is based on the fact that certain kinds of math problems are hard to solve. RSA is a one way function based on integer factoring, and DSA is a one way function based on discrete logs. However, an infinitely fast computer will be able to break the encryption. The trick is to make it take long enough that someone with finite computing resources cannot break it in a reasonable time. As key length increases linearly, key strength increases exponentially.

Key Distribution

In order for PGP to work, you have to *know* that Alice's public key belongs to Alice, and not an impostor posing as Alice. This problem is solved with key distribution. There are three methods of key distribution:

- Direct person to person
- Trusted third party (certifying authority) – Example: [Verisign](#).
- Web of trust

A web of trust is a combination of the first two methods. An example of the web of trust is a party: each person knows a couple of other people, who know a couple other people...and eventually everyone is indirectly known. In the web of trust, you rank people based on how they verify the identities of other people. The more signatures on your public key, the more collective trust you have.

This model could possibly be broken by impostor infiltration, and is generally used for personal applications rather than business applications.

10 Minute Break

GPG Demo

`gpg --help` or `man gpg` to get help.

`gpg --gen-key` to generate a key. The program will walk you through the process

- The longer the key, the stronger it is. However, some card readers will only support up to 1024 bit keys.
- Expiration is suggested, in case you ever lose your password.
- If you do lose your password, you cannot revoke your key...ever.
- Follow the prompts, enter your name, email, comments.
- **Note:** The key fingerprint is the hash of the key.

`gpg --gen-revoke <public key>` can be used to revoke your key in case it has been compromised. This will make a gpg public key block, which you can upload to whichever servers you are using to show that your key has been revoked.

`gpg --recv-key <key hash>` gets a key from the server.

`gpg --send-key <key hash>` sends a key to the server.

`sign` or `lsign` will sign (or locally sign) a public key.

`gpg --armor --encrypt -r email@host.net` will encrypt a message, and will resolve a public key from the email address. The armor option makes the output readable, rather than binary. Then you type your message. Alternately, you could pass `gpg` a file name.

`gpg --decrypt <paste message content>` will decrypt a message. Alternately, you could pass a file name.

`gpg --clearsign` will produce a gpg digital signature.

`gpg --decrypt` will decrypt a digital signature. If the message has been changed, the signatures will not match.

Questions

Q Why is it recommended to use separate signing/encryption keys?

A If you're using your signing key for a public key verification protocol like the following:

Provider	Direction	Verifier
name	→	
	←	random challenge
Secret Key(random challenge) = r	→	Public Key(r) =? random challenge

By the protocol above, the verifier can ask the prover to encrypt anything they like on their private key so they could induce the verifier to decrypt a message if their encryption and signing keys are identical. While the example is a little contrived, it demonstrates that it is good practice to avoid the issue completely by the relatively easy step of separating the keys.

General Comments

The concept of PGP encryption is reasonably confusing, at least for a new user. Check out the [Wikipedia page](#) for a better understanding with illustrations.

The question of meeting time and location was raised. Stautzenberger College has moved to a new location, and is no longer available as a meeting location. The proposed alternatives included the Toledo Public Library, and the University of Toledo. Based on the fact that UT has a Unix lab with a projector, it was decided to hold a few meetings there. Unfortunately, the downsides of this meeting location are that the projector is a bit fussy with certain hardware, and people have complained about the noise of the ventilation system.

The question of meeting times was unresolved, and has been moved to the TALUG website in the form of a poll. Hopefully this will shed some light on optimal meeting times.

Additionally, a PGP key-signing party was proposed, but no date was set. Perhaps this could be done in conjunction with a future meeting.

Asus Eee PC 701

Gary brought his new toy along to the TALUG meeting, an [Asus Eee PC 701](#). The Eee PC is an “ultra-portable laptop” weighing 2 pounds, and comes with a 900 MHz Intel Celeron-M processor, 512 MB ram, 4 GB solid state hard drive, and wired/wireless network cards. One of the cool things about the Eee PC is that it has a SD reader on the side, so the available hard drive space could be easily expanded. Pictures of the laptop can be found on the [TALUG website](#).

The Eee PC runs a modified Xandros distribution with a simplified user interface ([interactive demo](#)) utilizing tabs to categorize applications. Alternately, the laptop can be used with KDE as the desktop environment by simply pressing the power button and selecting KDE. All in all, it was a pretty slick laptop at a reasonable price.

After Meeting Activities

After the meeting, dinner was suggested. Craig Seeley recommended Ipoh, a Chinese restaurant on the corner of Door and Byrne, less than a mile away. Eleven people joined, and with good company and good food the PGP/Linux discussion continued until approximately 10:30 PM.