

Create Secure Web Forms with PGP/GPG

How to protect customer data and avoid falling victim to high-tech credit card thieves.

By Frank Rietta
March 1, 2004

During one recent e-mail virus epidemic, one of my merchant friends, discovered, to his horror, that a recently infected computer was broadcasting credit card numbers out to the Internet! The incident was devastating to his business, as he had to inform his customers that their private credit card information had been compromised. While this story involves several insecure applications and terrible security practices, I want to narrowly focus on one particular problem: the virus was able to catch the credit card numbers because he had received them from his website in an e-mail!

Some might suggest that the problem that this particular business was receiving on-line orders via e-mail. That is, he collected customer data over a "secure" website and then immediately rebroadcast that information through insecure e-mail. He would have been better off to have stored the order on the server to be later retrieved over an SSL encrypted website. Indeed, this is a good suggestion, but it too has two serious problems, firstly, is the server really trustworthy, and secondly, what happens if the data is ever compromised due to software or human error.

The real trick is to keep the data encrypted everywhere, that is, in transit from the customer, in storage on the server, and in transit to the merchant. An inexpensive way my merchant friend could have both remained secure and enjoyed the convenience of receiving his orders via e-mail.

Since the early 1990s, a little-known, inexpensive program called Pretty Good Privacy (PGP) has been available over the Internet. PGP and, the related, GnuPG (GPG) can be used to easily and extremely securely store sensitive data on disk and also e-mail it, with great confidence, across the network.

This article will show one way to collect data over an SSL encrypted website, format and encrypt the plain-text using a script and PGP/GPG, and then send the new cipher-text via normal e-mail to the recipient.

Prerequisites

- PGP is an e-mail encryption program, originally written by Phil Zimmerman, which is available for all UNIX-like systems (including Linux and FreeBSD), MacOS, and Windows. PGP is available free of charge for non-commercial users and for as little as \$50 (US) for

Create Secure Web Forms with PGP/GPG

DRAFT IV

commercial users. GnuPG (GPG) is a fully compatible implementation of PGP that is free for both commercial and non-commercial users. GPG ships by default with almost every distribution of Linux and is available for many other platforms.

Visit the following websites for more information or to acquire the PGP/GPG software:

- www.pgp.com - Commercial PGP
 - www.pgpi.org - PGP Source Code and Freeware Versions
 - www.gnupg.org - GNU Privacy Guard, free PGP compatible application.
- A basic understanding of encrypted websites (HTTPS using SSL).
 - A basic understanding of Unix shell commands. Examples in this tutorial assume that access is available to either Unix-like system, such as Linux, FreeBSD, Solaris, etc.
 - The code examples in this article are written in the PHP scripting language and make use of the free GPG running on a Linux server. However, the ideas readily transfer to other scripting languages and operating systems.

Those interested in implementing the encryption on a Windows server, may wish to take a look at the PGP SDK, which PGP Corporation sells through their website.

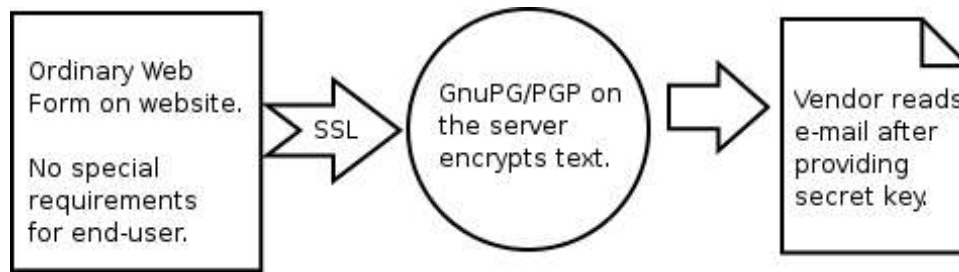
Visit the following websites for more information or to acquire software:

- www.php.net - The PHP Scripting Language

Outline

1. The Goal of Securely Collecting and Transmitting
2. Understanding PGP Cryptography
3. Preparing the Server to Encrypt
4. Collecting Web Form Data
5. Summary
6. References and Other Sources of Information
7. About the Author

1. The Goal of Securely Collecting and Transmitting



1. A customer goes to a website and fills out and submits an order form.
2. The server takes the order and e-mails it to the vendor, encrypted so that only the vendor can read the message.
3. The vendor receives an e-mail with the order, which he then decrypts by entering the password for his private encryption key.

PGP makes it easy to send private messages to anyone with the necessary software and keys. It can also be conveniently used to allow order information to be collected from a customer on-line and encoded in such a way that only the intended recipient (in this case the vendor) can read the message. Even if someone manages to intercept the message, he will not be able to read it since he does not have the vendors secret key.

Amazingly enough, this software has been available free of charge since 1991!

2. Understanding PGP Cryptography

Entire books have been written on cryptography. One good book for historical and basic technical understanding is *The Code Book*, by Simon Singh.

Phil Zimmerman wrote Pretty Good Privacy (PGP) so that ordinary computer users could have access to strong cryptography allowing messages to securely be sent without fear of prying eyes. PGP combines traditional cryptography with public key encoding.

The workings of PGP can be summarized through the story of Bob, Alice, and the nefarious Eve. Alice wishes to privately contact Bob, who has a publicly published PGP key. Alice uses her copy of PGP to encode the message in such a way that only Bob can read it with his private key. Eve is able to intercept Alice's letter while it is in transit, but she is unable to read the message because she does not have, and cannot practically derive, Bob's secret key. However, Bob is able to easily read the message once he receives it because he has his key.

One can have two keys, one that everyone in the world knows, and one that is kept private. As long as the private key remains private, everyone in the world can send secure messages, while none other than the proper recipient can read them.

Generate a Key Pair

A PGP key pair is needed before messages can be encrypted. PGP makes it easy to generate a new key pair, which is a public key to be provided to anyone needing to send private messages, and a secret key, which is necessary to read any message protected with the public key.

Keeping the Secret

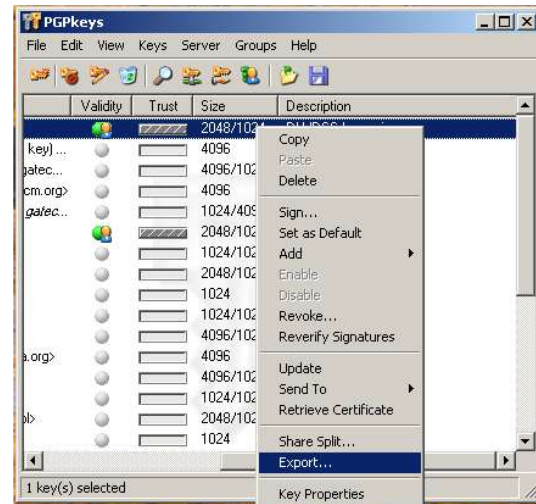
Once a key pair has been generated, it is important to keep the secret key private. PGP protects the key file with a pass-phrase. You need to be sure to protect the private key with a strong pass-phrase and keep it secret. You should also control which computers have your private key file. Never upload it to a public computer or post it on the Internet. Avoid placing it on your web server.

Publicizing a Public Key

The public key can be publicized on the Internet, printed in any newspaper, or handed to every single living human being in the world!

Everyone with a copy of the public key will be able to send you private correspondence and also verify documents that you cryptographically sign (which is a nice feature outside of the scope of this article).

Since the public key is designed to be published everywhere, this is not a security concern in placing it on a web server, which will allow a secure web form to be implemented.



The Export option in PGP's key manager.

3. Preparing the Server to Encrypt

Before you can configure the server to encrypt web forms, you will need a public key. If you do not already have a public key, refer back to section 2, and generate a new key. You should export the public key from PGP as an ASCII armor file.

Once you have your PGP ASCII armor file for the public key, the first step

Create Secure Web Forms with PGP/GPG DRAFT IV

is to configure GPG on the server. In my example, I am going to suppose that SSH access to a Unix server with GPG already installed and that CGI scripts run as my user.

First, I am going to log into the server through SSH and check to see if GPG is present, and if so, which version it is.

```
$ whereis gpg
gpg: /usr/bin/gpg /usr/share/man/man1/gpg.1.gz
$ /usr/bin/gpg --version
gpg (GnuPG) 1.0.7
Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
Home: ~/.gnupg
Supported algorithms:
Cipher: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA, ELG
Hash: MD5, SHA1, RIPEMD160
```

A temporary directory will be necessary for encryption as GPG will be provided with a text file to encrypt. While the generic system-wide temporary directory (usually /tmp) will work, it is a good idea to make a special directory which can have permissions set to exclude all other users from accessing any of the files used during the encryption process. The Unix permission flag 600 equates to the owner can read and write and all other users have no access. Here are the commands that will create a cipherTmp directory and prevent all users other than the owner from accessing any data held within:

```
$ mkdir cipherTmp
$ chmod 600 cipherTmp
```

I will now import the public key, which I exported as an ASCII armor file and uploaded to the server previously. In this example, my file is called orders_pub.asc.

```
$ gpg --import orders_pub.asc
...
gpg: key 4860E4C7: public key imported
gpg: Total number processed: 1
gpg:                imported: 1
```

The public key has been successfully imported and can be used to encrypt

Create Secure Web Forms with PGP/GPG DRAFT IV

messages. To test the ability to encrypt messages, create a text file with the message "Hello, World":

```
echo "Hello, World" > hi.txt
```

and then encrypt it to the imported key:

```
gpg -r orders@example.com -ae hi.txt
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: B26B67A3: There is no indication that this
key really belongs to the owner
2048g/B26B67A3 2001-10-19 "Order Box
<orders@example.com>"
...
It is NOT certain that the key belongs to its
owner.
If you *really* know what you are doing, you
may answer the next question with yes
Use this key anyway? Yes
```

I answer "Yes" to the prompt and now have a file called hi.txt.asc, which has the following text as the following cipher text contents:

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.0.7 (GNU/Linux)

hQIOA+C96q2xa2aJEaf/QmFODqidggTRUOS0q7SESSf/UEr
xcxD4Akxyqo/G7bWdKaSesOmoyEvrDvLwhjr4IfgTJzZbAY
aKMvnwewq7/pDyhi2v+H75vWiiO8MtlDo0Py70Lqs0bSKPE
4adn7MoPrrG3DG+8Fjg+6lKh+wmefwRMKqPK5DBgnNMl44u
VcgwTVFPSSWD/20FuWSmjKfsMnaCFb7/bsp+wL8vYI7cqF3
WwG7hgZHS3HpUTMLNMixwZZFWXWbaJ1zWpirTSajYEi9Z4Z
2W07Yc7vRvZcc5dGuVrfdAVNnKIXCDtHGKdLAv6PfWgxfG
6usTcXBOoY09LlHiyvYlGeaztFEKB6xIwf9E3Md3Ao/LTPs
pu3wAtUCgXzMo9YBxOuNHTRQalrW8yDDQXttl7uoC/7yJw
vrWz5z7ALu/Bgwem55Ui/+9G0xr4PYlbQ0FEJnOKBahaVlQ
4jVNaZoIuGhvrQWODlpKklT2e740fD83ptAG+7qY62iaqut
K5vLcAmLSqaX+fBRYv8gJyh0S09GUzf3pw+hayBY14t4qZf
IMXqGWVOMvRH+8fZnW+VEY2wADJGrLwSI0bUbs/IQ1mxKfQ
AHxm61wNRhSU63/P3e3vw/pv/XI1E8JnQVqTWS9iBSUe6V6
2NvoQpOQqzT0lip4h79ox07AjwXo4gKbdTHyg586f3o+/hy
skppYF+DKyRuXxN+ELhQss/qPvWSOezU8o5uZnW0bMCS+wH
HWvgc2fEq5I==4VXJ
-----END PGP MESSAGE-----
```

However, there is a problem, GPG does not trust the key I imported and therefore is going to prompt the user each time it is asked to encrypt something. This is not workable since there will be no one to answer "Yes"

Create Secure Web Forms with PGP/GPG

DRAFT IV

each time the web form script will call GPG. The solution is to sign the key, but I will first need a signing key on the server. For this purpose, I can generate a “throw away” RSA key in order to sign the public key.

```
$ gpg --gen-key
```

GPG will ask a series of questions, choose the following options:

- (5) RSA (sign only)
- Accept the default key size of 1024 bits.
- Specify that the key will expire in 1 day.
- Provide the name for the key something like, “Frank’s Website Signing Key”
- When prompted for a pass phrase, you can either specify one or leave it blank. I choose to leave it blank since this is a throw away key that will expire tomorrow.

Now, I can sign my public key with the new throwaway signing key.

```
$ gpg --edit-key
```

GPG will present an interactive prompt with a list of keys; in my case I see the following:

...

(1). Order Box <orders@example.com>

I issue the `lsign` command at the interactive prompt.

```
> lsign 1
```

Now the the key is signed, I can exit the interactive prompt with the `quit` command.

```
> quit
```

Now, I am going to delete the previously encrypted `hi.txt.asc` and reencrypt the original file.

```
$ rm hi.txt.asc
```

```
$ gpg -r orders@example.com -ae hi.txt
```

There were no errors and there was no prompt and GPG indeed created a new `hi.txt.asc`, which is encrypted as before.

5. Collecting Web Form Data

Secure, SSL websites are handled by existing web servers and browsers. It is quite nice that creating a secure, encrypted web form is as easy as creating a non-encrypted form! Simply put together the HTML pages and

Create Secure Web Forms with PGP/GPG DRAFT IV

the script to accept the submission and place them on a server with an HTTPS in the URL.

Since I have successfully setup GPG on the server and installed the public key for the orders box to the website user's key ring, I can now create a few test scripts.

The first file is an ordinary, everyday HTML form that takes a credit card name, credit card number, expiration date, and type of card. The form is set to post to exampleSubmit.php, which is the script, which will take and encrypt the data.

Listing 1: example.html

```
<form action="exampleSubmit.cgi" method="post">
<input name="nameOnCard" type="text">
<input name="cardNumber" type="text">
<input name="expirationDate" type="text">
<select name="typeOfCard">
<option selected>Visa/MasterCard</option>
<option>Americian Expresss</option>
</select>
<input type="submit" value="OK">
<input type="reset">
</form>
```

Once the HTML file is created, I am going to need to write up a script to take the form submit data, encrypt it, and then e-mail it to the orders mail box. The script does the following:

- Imports `pgpWrapper.inc`, which will be shown in detail next, in order to define the `pgpEncryptText` function.
- Creates a plain text string with all of the pertinent information from the POST data.
- Produces a cipher text string by running the plain text string through `pgpEncryptText`, specifying the key as the orders@example.com.
- E-mails the encrypted text, as the body, to orders@example.com.

IMPORTANT: GPG must be run as the user who owns the keyring, which is most likely not the default Apache user, "nobody." This is easily handled if the Apache SuExec option is

Create Secure Web Forms with PGP/GPG DRAFT IV

enabled on the server, if you are not sure if it is enabled check with the server administrator. Usually PHP scripts are run under the web server's user, that is "nobody," but CGI scripts are run as the owner. As long as the PHP CGI binary is available on the server, this will work if the script is named with the "cgi" extension instead of the traditional "php." In order for this to work, the path to the PHP CGI binary must be included on the first line of the script.

Listing 2: exampleSubmit.cgi

```
#!/usr/local/bin/php
<script language="PHP">
/* Import the pgp wrapper, shown later */
include "pgpWrapper.inc";
/*
 * Collect the order information as a single
 * string.
 */
$text = "New Credit Card Order\n"
    . "\nName: "      . $_POST["nameOnCard"]
    . "\nAcct Number: " . $_POST["cardNumber"]
    . "\nExp Date: "   . $_POST["expirationDate"]
    . "\nCard Type: "  . $_POST["typeOfCard"]
    . "\n";
/*
 * Now encrypt the text using the pgp wrapper.
 */
$secureText =
    pgpEncryptText("orders@example.com", $text);
/*
 * Now e-mail the encrypted order text
 */
mail ("orders@example.com",
      "New Encrypted Order",
      $text);
</script>
Thank you for the order!
```

And finally, the pgpWrapper.inc script, which is designed to be included in other PHP scripts where text encryption is required. The script defines a function called pgpEncryptText that follows the following process:

- Take a plain text string and a PGP key to which it should be ciphered.
- Generate a temporary file to which the plain text is written.

Create Secure Web Forms with PGP/GPG DRAFT IV

- Run PGP/GPG on the plain text file in order to encrypt it, a new cipher text file will be created with the .asc extension, which is an ASCII armor file.
- Immediately delete the plain text file.
- Read the cipher text file into a new string.
- Immediately delete the cipher text file.
- Return the new cipher text to the calling function.

Security experts assert that this process is inherently flawed as the plain text has to be written to disk for a period of time. That a nefarious user on the system could read the plain text before it is deleted or that a power failure would cause the script to fail to run to completion leaving a plain text file available on disk. A project called GPGME is working to solve the problem and allow programs to work with GPG without writing files to disk.

The above being said, working with temporary files is the simplest way to set things up. One way to mitigate some of the dangers of writing files to the disk by creating a cipherTmp directory, which only the website user can access (chmod 600), as a working space for the pgpWrapper script and GPG. It is still as important as before to practice good account security, such as keeping a strong password and never using non-encrypted means (such as FTP) to access the server.

Listing 2: pgpWrapper.inc

```
<script language="PHP">
function pgpEncryptText($key, $text) {
/*
 * CHANGE THIS: The location of the GPG settings
 * files, for most cases, just replace wwwuser
 * with the proper username.
 */
$gpgSettingsDirectory = "/home/wwwuser/.gnupg";

/*
 * CHANGE THIS: The location of the owner-only
 * temp file directory for GPG to use. The
 * directory should be chmod 600, which will
 * prevent other users from being able to
 * access the file while it exists.
 */
$workDirectory = "/home/wwwuser/cipherTmp";

/*
 * Set the Environment Variable, which GPG will
 * use to determine where its settings are
 * stored.
 */
putenv("GNUPGHOME=$gpgSettingsDirectory");
```

Create Secure Web Forms with PGP/GPG

DRAFT IV

```
/*
 * Get a name for a temp file in the
 * work directory.
 */
$STempFile = tempnam($workDirectory, "");

/*
 * Get the name of the encrypted file, which
 * will need to be read into a string. The
 * name is the temp file with .asc appended.
 */
$STempEncFile = $STempFile . ".asc";

/*
 * Write the text to the temp file.
 */
$NFile = fopen($STempFile, "w");
fwrite($NFile, $text);
fclose($NFile);

// If the encrypted file exists, delete it.
if( is_file($STempEncFile) )
    unlink($STempEncFile);

/*
 * Prepare the command to encrypt the text in
 * ASCII armor format, which will later to
 * returned to the caller.
 */
$SPGPCCommand =
    "/usr/bin/gpg -r $key -ae $STempFile";

// Run the command!
system($SPGPCCommand);

// Delete the plaintext file
if( is_file($STempFile) )
    unlink($STempFile);

/*
 * If the encrypted file exists,
 * read it back into a string and then
 * delete it from disk.
 */
if( is_file($STempEncFile) )
{
    // Read the encrypted text
    $aEncryptedText = file($STempEncFile);

    // Turn the array into a string
    $sEncryptedText =
        implode("", $aEncryptedText);

    // Delete the temporary file.
    unlink($STempEncFile);
} else {
    $sEncryptedText =
```

Create Secure Web Forms with PGP/GPG

DRAFT IV

```
        "Error, GPG did not create ciphertext." ;
    } // end if encoded file exists.
    return $sEncryptedText;
} // end function pgpEncryptText
</script>
```

5. Summary

Creating really secure, encrypted, web forms is not only inexpensive, but easy using PGP/GPG and a few scripts. In this article, one way to accomplish a cryptographically secure e-mail form on any server with PHP and GPG, which are both freely available.

On-line security has become more relevant as companies, both small and large, start to conduct more transactions on-line. Customers are more technically savvy than ever and want to be sure that their private information is protected. Fortunately, with a little know-how, even the smallest home businesses can afford really strong protection for their data.

Even a small business using a Windows desktop and a website hosting account can procure the tools necessary to utilize strong cryptography for as little as \$50.00 (US), which is the cost of a single commercial license of the PGP for Windows.

6. References and Other Sources of Information

- PGP can be found at www.pgp.com and also www.pgpi.org.
- GPG can be found at www.gnupg.org.
- The PHP scripting language can be found at www.php.net.
- Apache has a facility, called SuExec, to allow CGI programs to run as a user other than the default, "nobody." Many web hosting providers support this feature and documentation can be found at <http://httpd.apache.org/docs/suexec.html>.
- KGPG is a KDE-based front-end for GPG which allows for basic key management and encrypting/decrypting text from the clipboard. It is available at <http://devel-home.kde.org/~kgpg/>.
- *The Code Book*, Simon Singh (1999).
- *Pgp: Pretty Good Privacy*, Simson Garfinkel (1994)
- *Crypto: How the Code Rebels Beat the Government Saving Privacy in the Digital Age*, Steven Levy (2002)

7. About the Author

Frank Rietta has been working with PGP and Linux/BSD for more than three years. He is a developer member of the Association for Shareware Professionals and has served as the President of the Georgia Tech Student Chapter of the Association for Computing Machinery (ACM). Frank is a software developer and the owner of Rietta Solutions, which provides various data hosting services.

When cryptography is outlawed, bayl bhgynjf juyy unir cevinpl.