

# Creating a secure Peer to Peer connection between known Parties

***Abstract:** What follows is a proposal for a protocol to establish a secure connection between two peers. It may be extrapolated to build a network of authorized peers. This protocol is only one part in what would be a larger system; where this protocol leaves off an application specific protocol must then be applied to implement the actual business function.*

Creating a secure Peer to Peer connection between known Parties.....	1
1. Introduction.....	1
1.1 Encryption.....	2
1.2 Exchange of public keys.....	3
1.3 The Human factor.....	3
1.4 Multiple Session Keys.....	4
1.5 What's encrypted?.....	5
1.6 Key Expiry.....	5
2. The Meta Protocol.....	5
2.1 Implementation.....	5
3. SecureConn Protocol 1.0.....	6
3.1 Implementation – Determining Algorithms.....	6
3.2 Implementation - Exchanging Public Keys.....	7
3.3 Implementation – Session Key Exchange.....	8
3.4 Protocol Completion.....	9
4 Conclusion.....	9

## 1. Introduction

Since their creation Peer to Peer communications have primarily been left to the domain of public file swapping. However there are many business applications that could benefit from Peer to Peer functionality. One concern for businesses entering this communication arena is the issue of security. A Peer to Peer business application would be transporting inventory records, pricing information, financial data and research documents, all of these data need to be keep private. As a result, before these business applications can take advantage of the benefits of Peer to Peer systems, the security of private information must be assured.

In this document I will introduce a protocol seeks to establish an easy method by which two peers can establish a secure connection. An application using this protocol could establish secure communications, opening the doors of Peer to Peer functionality of business. For example, using this protocol a disturbed inventory control system could be developed without the need for a centralized server. The same security could be achieved

by creating a secure network, such as a Virtual Private Network (VPN). However a VPN solution is limiting because it requires that anyone using the application first establish such a secure network. In addition, Technologies such as<sup>1</sup> are being developed to provide transport level encryption for applications. However, this solution is not yet stable and requires an external setup. The peer to peer protocol in this document avoids the drawbacks of both a VPN and IPSec by integrating the encryption directly into the product. As a result, external set-up of separate secure networks is no longer required

## 1.1 Encryption

In order to build a secure network, all communication must be encrypted before it is transported. Having to encrypt all the communication data can be cumbersome because of the sheer volume of the data. As a result the speed of the encryption process is important. This protocol will employ symmetric encryption, as finding have shown that it is traditionally 100 times faster than asymmetric in software implementations (SCHNEIER 1996). However, asymmetric encryption is required to facilitate the initial connections in order to relieve the operators from having to transfer keys securely between one another. Like almost all other implementations of asymmetric encryptions, this protocol will rely on asymmetric to encrypt randomly generated sessions keys. The randomly generated session keys will in turn be used to symmetrically encrypt our data transmissions.

Another possible approach to security would be the use of passwords or pass phrases to encrypt session keys. This protocol will not rely on passwords or pass phrases because it has been shown that the speed of computers in cracking passwords is quickly outpacing the human ability to remember passwords or pass phrases with enough entropy (Schneier, 1963). Subsequently, Schneier (1963) suggests that the easier a password is to remember the less secure it is. Sadly, it has become evident that in many cases passwords are now useless.

Having determined the need for both a symmetric and asymmetric encryption we must next determine which algorithms to use. All current asymmetric encryption schemes rely on problems which are assumed to be "Hard." However, advances in number theory may prove these assumptions wrong. An advance of this type would make any encryption scheme based on that problem worthless. Any application which had selected that scheme as it's only means of encryption would also be rendered useless. In a more likely scenario, a faster algorithm may become accepted. In addition, a flaw may be discovered in a selected implementation of an algorithm. Consequently selecting any particular algorithm for a long term encryption scheme is deadly. It is for this reason, that this protocol provides a mechanism for Peers to dynamically determine which algorithms to use for any given session. Further, applications using this protocol should be built with a system for interchanging implementations of algorithms, and a method for giving these implementations definitive names. Lastly, the protocol also establishes a

---

<sup>1</sup> IP Security Protocol  
<http://www.ietf.org/html.charters/ipsec-charter.html>

mechanism for preference amongst algorithms, so secure but slow algorithms can serve as back up in the case that a peer does not support the faster algorithm.

In order to for two peers to communicate they must implement at least one symmetric and one asymmetric algorithm in common. For this reason the protocol suggests until further developments render these choices unreasonable, that all implementations support ElGamal (ELGAMAL 1985) for asymmetric encryption and CAST-128 (ADAMS 1997) for symmetric encryption. These are currently used in PGP<sup>2</sup> and have been determined to be good choices by people with far greater understanding of the subject than my own.

## 1.2 Exchange of public keys

Before two Peers may communicate effectively they must exchange public keys. In some other asymmetric applications this is done through a trusted third party, using key servers. The use of key servers provides two significant advantages. First, key servers allow the exchange of keys without the need for the two parties to communicate directly. Second, key servers allow some degree of trust that the particular key received relates to the requested person. This is particularly necessary when developing a system where the two parties may not reasonably be expected to communicate directly. Neither of these advantages is required by an application using this protocol as the two peers will be communicating directly. As such, the beginning of the protocol determines the procedure for determining public keys for the chosen algorithm. When one peer first contacts another, they determine which asymmetric algorithm to use and then exchange public keys (these keys can be generated separately for each peer).

## 1.3 The Human factor

The problem of authenticating the exchanged keys still needs to be addressed. Without some means of authenticating the exchanged keys there is no protection from a man-in-the-middle attack. Admittedly, most applications of the protocol will use direct TCP connections so anyone mounting such an attack would have to spoof IP addresses or intercept and modify packets on the fly. However, a good protocol should address these issues.

After public keys have been exchanged, the protocol calls for a disconnection. Applications should only connect again once the received public key has either been authenticated or discredited. Because developers cannot rely on proper use from their users I suggest that proper use be forced. Here is an example of a typical key authentication question:

"If this fingerprint valid? It must be  
validated before communication can continue"

---

<sup>2</sup> [www.pgpi.com](http://www.pgpi.com)

Users presented with such a question will instinctively select yes, destroying the security provided by the protocol. Here is an example of a method more likely to elicit a secure use:

```
"Please contact the person operating the peer at
IP-ADDRESS. Please have them read the finger
print they received and type it into the
following box. They will need a similar finger
print from you, please read to them the following
finger print"
```

The users can exchange the fingerprints of the keys they sent, the applications can then verify that the keys have not been tampered with.

## 1.4 Multiple Session Keys

It has been established that asymmetric encryption will be used to encrypt randomly generated session keys. Unlike most systems, the protocol suggests the use two (or more) session keys for each session. One will be used for general protocol information while the others will be used for data transmission. The reason for multiple keys is threat management. Due to their nature protocols are extremely repetitive and predictable, and as such are susceptible to known-plaintext attacks. This means that over time the session key encrypting the protocol data could become compromised. The primary response to this concern is to select a sufficiently secure algorithm and corresponding key size. In addition, session keys should expire and be replaced; the protocol offers a method of doing this. Even with these measures it is irresponsible to overlook this potentially fatal flaw.

Fortunately, if a session key is compromised it does not necessarily mean that the private key with which it was transmitted is in jeopardy. To secure this, when session keys are transmitted, they are appended with another random string which is discarded. Using this method, if a session key is discovered known-plaintext may not be applied to the private key.

So with the protocol session key compromised but the private key secure we need only concern ourselves with what is secured with this session key. Although a lot of valuable information can often be discerned from the protocol information, the truly important information will often be contained in the data that is exchanged. This data are much more likely to have high entropy and thus be effectively impervious to known-plaintext attacks. It is for these reasons, that a second session key is generated to encrypt the data separate from the protocol. The ability exists to generate additional session keys for any other reasons; the uses of these would be determined by the specific application.

## 1.5 What's encrypted?

With a private key and multiple session keys the question of what to encrypt arises. Almost all of this protocol is conducted in plaintext. In addition, all communication before the implementation of this protocol is also unencrypted. However, all communication following the protocol should be transferred encrypted using one of the session keys. A notable exception to this rule is if regression to this protocol is necessary. For example when a session key expires. In such a case communication similarly regresses to plaintext.

## 1.6 Key Expiry

It has been established in section 1.4 that the protocol session key is open to known plain-text attack. In order to further reduce the amount of possibly compromised information session keys should expire after a select length of time. How long a key is to remain valid is left up to the application level protocol which will be using the keys. It is suggested that keys remain valid for no longer than one to two days. It is also important to note that a known plain-text attack on a secure algorithm will take most attackers more than this length of time to compromise a key.

## 2. The Meta Protocol

This section of the protocol isn't specific to this implementation. It is used to provide room for changes in the protocol. This or something like it should be applied before any communication protocol. Its job is to simply agree upon a common implementation of the real protocol. As such, it designed to be as "bare bones" as possible, moving all higher functionality to the real protocol.

### 2.1 Implementation

For the purposes of demonstration, the peer initiating the connection shall be called the client. Whereas the peer receiving the connection shall be the server.

The client initiates the connection by suggesting a protocol:

CLIENT: SecureConn 2.0

The client should suggest its preferred method of communication. If the server can speak this protocol then it should respond with the same:

SERVER: SecureConn 2.0

Upon confirming a protocol, all future communications follow the agreed upon protocol. If however the server does not understand this protocol, it should respond with a suggestion of its own.

SERVER: SecureConn 1.0

This pattern will repeat until either a protocol is accepted, or one peer runs out of protocols to suggest. If either peer has no more protocols to suggest it replies:

PEER: NONE

Following this message both peers should break the connection. No further communication is possible until one peer learns a new protocol.

### **3. SecureConn Protocol 1.0**

Discussion of the reasons behind the protocol is covered in section 1. This section of the protocol is followed immediately after selecting “SecureConn 1.0” using the Meta protocol.

#### **3.1 Implementation – Determining Algorithms**

For the purposes of demonstration the peer which suggested this protocol in the Meta protocol shall be called the client. Whereas the peer who confirmed the protocol shall be the server.

The same method of suggestion and confirmation used by the Meta protocol is used to determine an accepted set of algorithms. The asymmetric algorithm is decided upon first, and the client is the first to suggest an algorithm.

CLIENT: ASYM: ElGamal 1024

The server may then either accept this algorithm, suggest an alternative or declare communication is impossible. To accept, the algorithm name is simply repeated:

SERVER: ASYM: ElGamal 1024

To suggest an alternative algorithm:

SERVER: ASYM: ElGamal 4096

In this example, the server is requiring a higher level of encryption. It is possible, that the client may have only suggested the lower version in interests of speed, but may be willing to communicate at the more secure level. Finally, the server may determine that no algorithm may be mutually decided upon:

SERVER: ASYM: NONE

Following this message both peers should break connection.

If an asymmetric algorithm is decided upon, then the process is repeated for the symmetric algorithm. The first suggestion is sent by the peer who suggested the accepted symmetric algorithm (called client from now on).

CLIENT: SYM: CAST-128

This is replied to with:

SERVER: SYM: CAST-128  
(to accept)

Or:

SERVER: SYM: AES 128  
(to suggest an alternative)

Or:

SERVER: SYM: NONE  
(to declare no mutual algorithm)

Again, if no algorithm can be decided upon both peers disconnect. If an algorithm is decided upon, then the protocol progresses with key exchange.

### 3.2 Implementation - Exchanging Public Keys

This section of the protocol is designed to exchange public keys between the connecting peers. A peer may create a separate private/public key pair for each peer with which it communicates (based on IP or something similar). Alternatively, a peer may use the same pair for all communications. A variation of the protocol may be developed to skip this step for connections that have been established in the past. However such a variation should be presented as different in the Meta protocol (SecureConn 1.1 for example). In addition, any such implementation should be able to revert to the normal protocol.

Again, Client shall refer to the peer which suggested the accepted algorithm from the last section. The client begins with transferring its public key:

CLIENT: ASYM-KEY: *Client Public Key*

The exact format of the key is to be determined by the specific asymmetric algorithm chosen, however most will transmit in ASCII Hexadecimal. The server responds with its key:

SERVER: ASYM-KEY: *Server Public Key*

Both peers now compare these keys to their internal key ring. If the client has not seen this key before or its operator has not authenticated it, then the client will respond with:

CLIENT: ASYM-KEY REJECTED – Not authorized

After this message both peers should disconnect. If however the client has seen this key before, and its operator has confirmed its validity then the client responds with:

CLIENT: ASYM-KEY ACCEPTED

Upon receiving this message, the server sends a similar set of messages. If the key is not known or authorized the following message should be sent.

SERVER: ASYM-KEY REJECTED – Not authorized

Or if the key is known and valid it responds with:

SERVER: ASYM-KEY ACCEPTED

At this stage the private keys are established and the infrastructure to send session keys is in place. If a peer ever wishes to change its private/public key pair, the peer should disconnect and reconnect this time sending the new public key. Doing this will cause the connection to be put on hold until the new key can be validated.

### 3.3 Implementation – Session Key Exchange

This section describes the development and exchange of a session key. This section is completed once, immediately following the exchange and verification of public keys in order to create and exchange the protocol session key. This section is then repeated in order to produce and exchange the first data session key. The reason for the use of multiple session keys is covered in section 1.4.

For this section, Client refers to the peer to last send their public key. The client remains the same for both rounds to session key exchange.

The client begins by creating two random numbers, the length of which is determined by the key size of the agreed upon symmetric encryption algorithm. The first number is the session key to be exchanged. The second number is a salt which is appended to the key in order to prevent a known plain-text attack against the asymmetric keys in the event the session key is compromised. The client should then encrypt the two numbers with the server's public key.

CLIENT: SYM-KEY-X:  $E_{Server}(Sessionkey\ RandomSalt)$

X represents the number of the numerical index of the session key being exchanged. The protocol key is always index 1. The first data key is always index 2. Other keys should have incremental numbers. If a key is to be replaced (because it has expired) the same index number should be used.

If the server cannot understand the message for any reason it may respond with:

SERVER: SYM-KEY-X: ERROR

Upon receiving this message both peers should drop the connection. If the Server understands the message then it should respond with:

SERVER: SYM-KEY-X: ACCEPTED

After the first completion of this section of the protocol, it is immediately repeated creating the first data key.

### **3.4 Protocol Completion**

After the exchange of session keys, the protocol is complete. Secure communication may begin using an application specific protocol. If a session key expires, (see section 1.6 for suggestions of key lifetimes) the application protocol may regress to the SecureConn protocol to repeat section 3.3 Session Key Exchange. This section may be repeated one or more times to recreate session keys as needed. If the private/public key pairs need to be refreshed, the connection should be dropped and re-initialized.

## **4 Conclusion**

I created this protocol to fill a void. I have been looking for an established protocol that Peer to Peer applications could use to determine algorithms and exchange keys. I believe it is this void which has kept many business applications from using the functionality that Peer to Peer networks can provide. In most applications a company would create a proprietary protocol and rely on the talent of their staff. Unfortunately, security is hard and it is very easy to make a fatal mistake. Even worse, the first indication that something was wrong in a proprietary protocol is the compromise of data. In an attempt to avoid the pitfalls many others have suffered, I am publishing this document and requesting feedback. Please let me know if you locate something that I have overlooked or apparently don't understand. I can be reached at [zelus@san.rr.com](mailto:zelus@san.rr.com). Thank you for taking the time to review this protocol.

- B. Schneier, "Applied Cryptography, Second Edition", Wiley, 1996.
- B. Schneier, "Secrets and Lies: digital security in a networked world", Wiley 2000
- C. Adams, "The CAST-128 Encryption Algorithm", May 1997
- J. Benz, "PGP: A Hybrid Solution", SANS Institute 2001
- S. Simpson, "PGP DH vs. RSA FAQ", Sam Simpson 1999
- T. ElGamal, "ElGamalEnc", ElGamal 1984
- T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete \ Logarithms," IEEE Transactions on Information Theory, v.IT-31, n. 4,