Project 3

Due June 2, 2016 at 11:55 PM

This project specification is subject to change. You may work alone or with a partner for this project. For those who choose to work with a partner, the portions labeled as Extra Credit must be completed in order to receive full credit. No extra credit will be available to those working with a partner. You must implement this project in C/C++. You will be expanding upon project 2 peer-to-peer application. Project 2 does not encrypt, nor does it authenticate its communications. You will be requesting/receiving authenticated public keys for individual users. You will be encrypting the TCP communications with the use of an encryption library. Submit a gzipped tar file of your source code, makefile, and a readme file.

Additional options

The application syntax is: p2pim [options]

-ap port — Specifies the UDP port the trust anchor receives its requests from, if not specified it is assumed to be 50552.

-ah host[:port] — Specifies a host and optionally the port in which to send unicast UDP authentication requests. Only one host may be specified on the command line. If the port is specified, the –ap option is ignored. If the port option is not specified the –ap option is valid.

Modifications to UDP Discovery Messaging

The UDP discovery messaging remains the same for discovering clients; however, messages for authentication have been added. The authentication requests are sent to the trust anchor if specified or a trust anchor that has been discovered. If no trust anchor has been discovered, authentication requests are broadcast. Authentication requests that do not receive responses within a time (use the same timeout mechanism as for project 2) are retransmitted. The authentication process starts with the application generating a non-zero random 32-bit value (use lower 32-bits of GenerateRandomValue function in provided encryption library). The value is the secret number that is then encrypted using the trust anchor keys (P2PI_TRUST_E and P2PI_TRUST_N). The trust anchor will be the only program that can decrypt the secret number using its private key. The trust anchor will lookup the user in its database of keys, and reply with the public key (and modulus) if one exists. If the user does not exist in its database, the keys will be zero. The secret number will be encrypted by the trust anchor's private key so that it can be decrypted using the public key. The checksum is the 32-bit ones complement of the secret number, username, public key and modulus. See the Useful Information section for more details on keys and provided source/applications. The client should send Request Authenticated Key Messages to the trust anchors' authentication port (default 50552) from its UDP Discovery Port (default 50550). The peer-to-peer client will not need to open and bind to the authentication port, but will reuse the existing UDP port.

Request Authenticated Key Message:

Size	4				2	8	n
Description	Signature				Type	Secret Number	Username
Value	P	2	P	I	0010	Public Encrypted	USER\0

Authenticated Key Reply Message:

Size	4				2	8	n	8	8	8
Description	Signature				Type	Secret Number	Username	Public Key	Modulus	Checksum
Value	P	2	P	I	0011	Public Encr	USER\0	*E*	*N*	Public Encr

Modifications to TCP Client Messaging

The TCP communication will be encrypted using public key encryption of a session key. Communication will be established with the requestor sending the Establish Encrypted Communication Message. The Public Encryption Key and Modulus Key (known as *e* and *n* in StringToPublicNED function) will be sent to the receiver. Even though these may be known from the trust anchor, they are sent to establish an encrypted session in the case where a trust anchor cannot be found. The receiver will randomly generate a 64-bit sequence number and will encrypt the high 32-bit and low 32-bit values using the public encryption keys. The encrypted sequence key will then be sent back to the requestor in the Accept Encrypted Communication Message.

Both parties will use this sequence number as a session key for encrypting further communications in the Encrypted Data Chunk Message. The first Encrypted Data Chunk Message sent by the requestor will use the sequence number plus one, and each subsequent Encrypted Data Chunk Message will increment the sequence number by one. The first Encrypted Data Chunk Message sent by the receiver will use the sequence number minus one, and each subsequent Encrypted Data Chunk Message will decrement the sequence number by one. The TCP messages used in project 2 will be encapsulated and encrypted inside Encrypted Data Chunk Messages, with no changes to the message contents; however, the Signature is omitted, and the message types have changed as noted below. Messages that do not perfectly align to a 64 byte boundary are padded prior to encryption with “random” bytes.

A new “Dummy” message has been added to help obscure real communication. The Dummy Message can be sent at any time between messages, and should be ignored by receiving application. The Dummy Message should not be sent at a regular interval, but at a random interval.

The user should be notified when encrypted communication is attempting to be established with either an unauthenticated user, or if the provided keys do not match the authenticated information. If a choice is given to the user to reject unauthenticated connections, the Discontinue Communication Message should be sent as a reply.

Establish Encrypted Communication Message:

Size	4				2	n	8	8
Description	Signature				Type	Username	Public Key	Modulus
Value	P	2	P	I	000B	USER\0	Generated Key	Generated Key

Accept Encrypted Communication Message:

Size	4				2	8	8
Description	Signature				Type	Sequence High	Sequence Low
Value	P	2	P	I	000C	Public Encrypted	Public Encrypted

Encrypted Data Chunk Message:

Size	4				2	64
Description	Signature				Type	Payload
Value	P	2	P	I	000D	Session Encrypted

Message Type Changes:

Description	Original Type	Encrypted Type
Establish Communication Message	0004	5555
Accept Communication Message	0005	AAAA
User Unavailable Message	0006	FF00
Request User List Message	0007	00FF
User List Reply Message	0008	5A5A
Data Message	0009	A5A5
Discontinue Communication Message	000A	F0F0
Dummy Message	N/A	0F0F

Dummy Message:

Size	2	62
Description	Type	Dummy Data
Value	0F0F	Random Data

Extra Credit

For extra credit add the ability to send a file from one client to another. The sender sends a File Transfer Offer Message when a file is being offered for transfer. The receiver client should prompt the user asking if the file should be received, when accepted or rejected the appropriate File Transfer Accept/Reject Message should be sent back. Data should then be transferred using File Data Messages, one File Data Message should be sent for every Encrypted Data Chunk Message. This limits the amount of data that can be transferred per message to 50 bytes. Data Messages, those being sent for instant messages should be allowed to be sent during the file transfer. When implementing the file transfer, it may be necessary to poll on the event POLL_OUT, otherwise the application may block on a write and become unresponsive.

File Transfer Offer Message:

Size	2	8	n
Description	Type	File Size	Filename
Value	F5A0	File Size	FILENAME\0

File Transfer Accept/Reject Message:

Size	2	2
Description	Type	Accept/Reject
Value	05AF	Reject = 0000, Accept = 0001

File Data Message:

Size	2	8	4	Data Size
Description	Type	Data Offset	Data Size	Data
Value	50AF	File Offset	Data Size	File Data

The inherent overhead of the file transfer is obvious, for improved performance, a Jumbo Encrypted Data Chunk Message can be added. The Jumbo Encrypted Data Chunk has 4KB of payload, messages that do not align to 4KB are padded prior to encryption with “random” bytes. The File Data Message can be increased in Data Size to 4,082 with the Jumbo Encrypted Data Chunk Message. The Jumbo Encrypted Data Chunk Messages can be used for any message type, and can be used randomly to hide the size of the true messages.

Jumbo Encrypted Data Chunk Message:

Size	4				2	4,096
Description	Signature				Type	Payload
Value	P	2	P	I	000E	Session Encrypted

Useful Information

In order to implement this project an encryption library has been provided. The encryption library is in the form of a C++ and header file that are both available on smartsite. In addition a program called p2pisetauth and p2pitrust have been provided to create authentication information and act as trust anchor respectively. The p2pisetauth program creates the public keys from username and password. In order to get the p2pitrust in /home/cjnitta/ecs152b to properly authenticate you on the CSIF, the p2pisetauth in /home/cjnitta/ecs152b should be run on the CSIF.

void StringToPublicNED(const char *str, uint64_t &n, uint64_t &e, uint64_t &d);
StringToPublicNED converts a string *str* into a public encryption key pairs *n*, *e* and *d* where *n* is the modulus, *e* is the encryption key (or public key), and *d* is the decryption key (or private key). This function can be used to generate keys from the string of username:password, where the username is appended with a colon and then appended with the password. This allows the program to prompt the user for a password and generate keys on the fly. This username and password should match the previously setup authentication with p2pisetauth.

void PublicEncryptDecrypt(uint64_t &data, uint64_t e_or_d, uint64_t n);
PublicEncryptDecrypt encrypts and/or decrypts data using the public key (e) or private key (d) and the modulus (n).

void GenerateRandomString(uint8_t *str, uint32_t length, uint64_t sequence);
GenerateRandomString creates a pseudo random string of bits *length* bytes long using sequence number *sequence*. This function is useful for padding and dummy data.

void PrivateEncryptDecrypt(uint8_t *str, uint32_t length, uint64_t sequence);
PrivateEncryptDecrypt encrypts/decrypts a string a bytes *length* long using sequence number *sequence*. This function essentially XORs the data with the result of a call to GenerateRandomString with the same sequence number.

uint64_t GenerateRandomValue();
GenerateRandomValue generates a 64-bit random value based on the hostname and time. This value should not repeat from run to run on the same machine.

uint32_t AuthenticationChecksum(uint32_t sec, const char *str, uint64_t e, uint64_t n);
AuthenticationChecksum generates the ones complement checksum for the Authenticated Key Reply Message.