CryptalkFUN

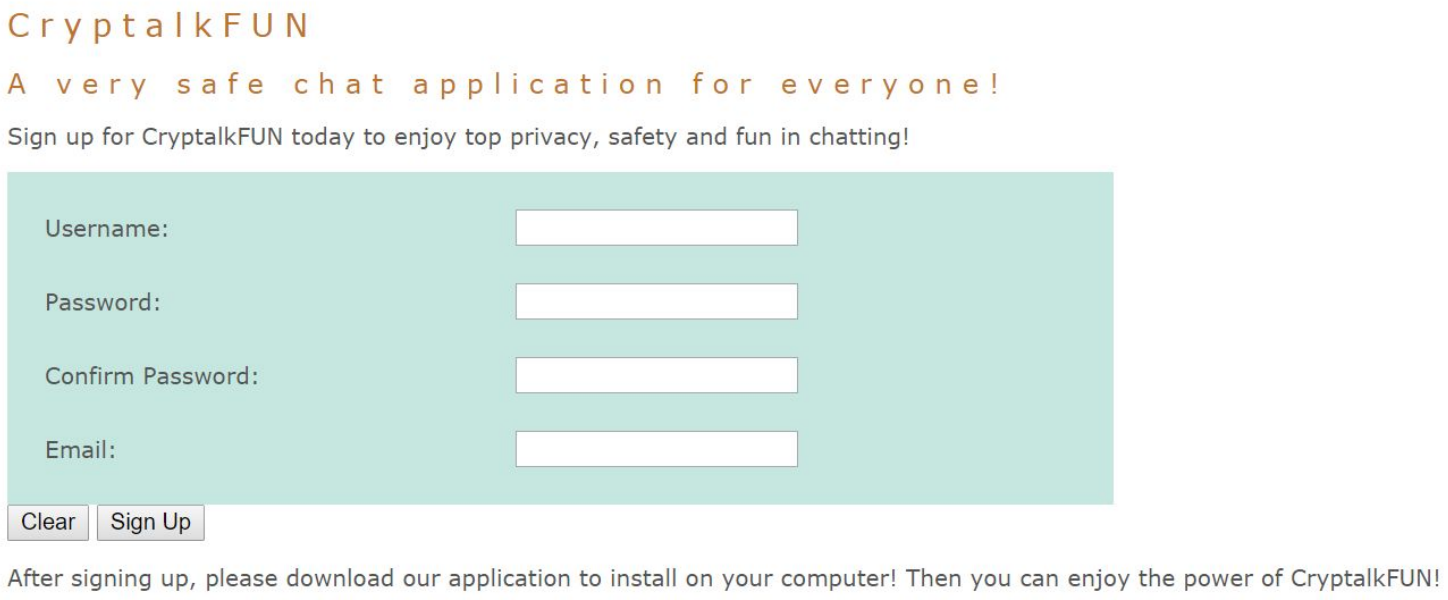
Che Wang, Liming Luo

Instant Messaging/Email

- Motivation: secure communication between two parties over a chat/email application
- Threats (Appendix A)
  - eavesdropping
  - data modification
  - replay attacks

Solution: end-to-end encryption

Application-Sign Up on Web


CryptalkFUN

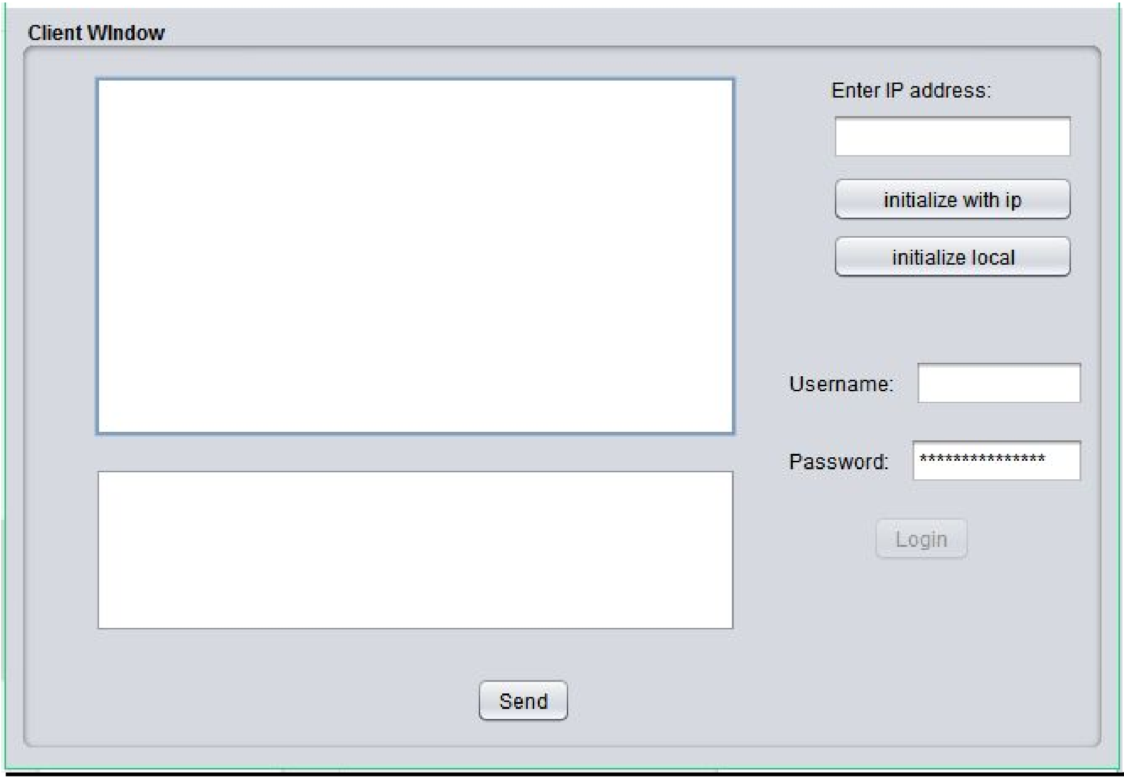
A very safe chat application for everyone!

Sign up for CryptalkFUN today to enjoy top privacy, safety and fun in chatting!

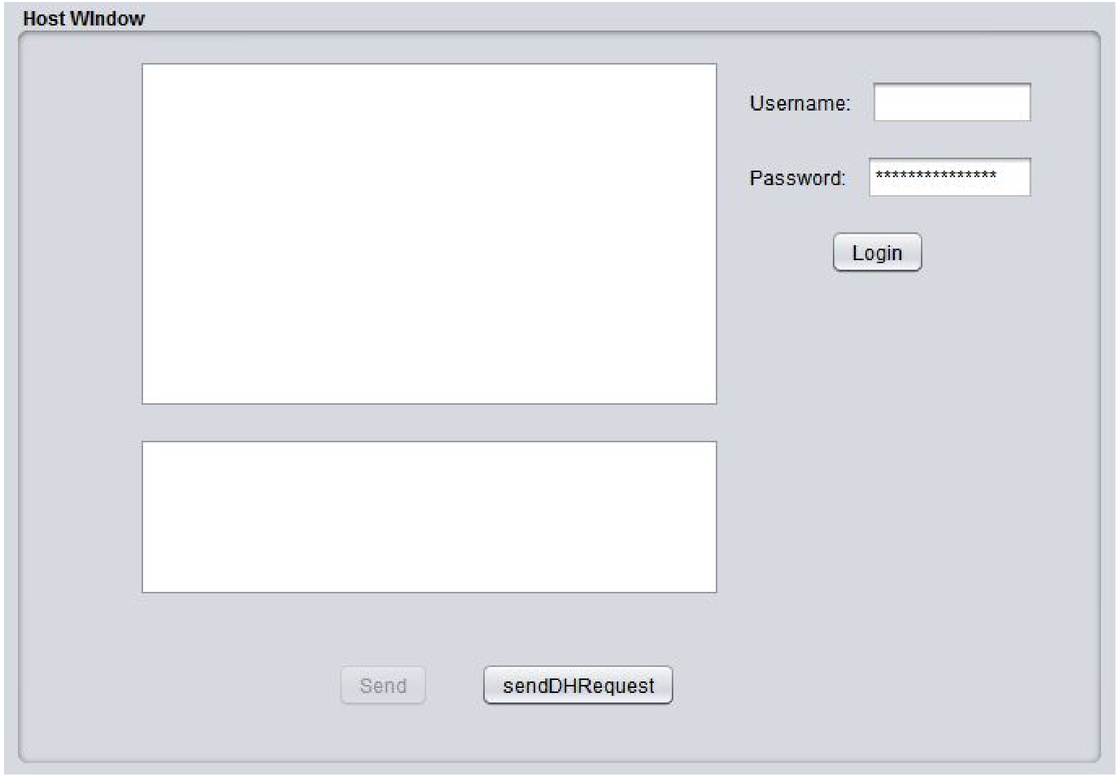
Username:	
Password:	
Confirm Password:	
Email:	

After signing up, please download our application to install on your computer! Then you can enjoy the power of CryptalkFUN!

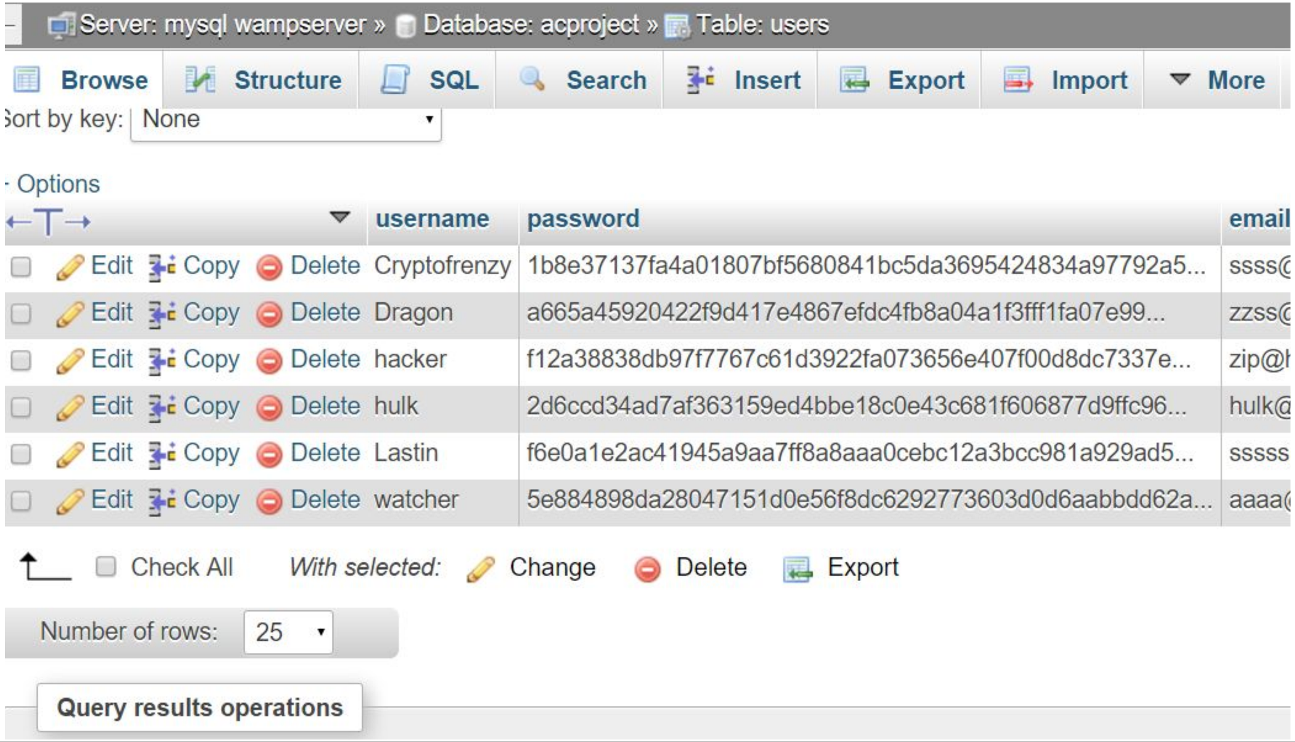
Application-ClientA



Application-ClientB (Host)



Application-User data in Database



Server: mysql wampserver » Database: acproject » Table: users

Browse | Structure | SQL | Search | Insert | Export | Import | More

Sort by key: None

Options

			username	password	email
Edit	Copy	Delete	Cryptofrenzy	1b8e37137fa4a01807bf5680841bc5da3695424834a97792a5...	ssss@
Edit	Copy	Delete	Dragon	a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e99...	zzss@
Edit	Copy	Delete	hacker	f12a38838db97f7767c61d3922fa073656e407f00d8dc7337e...	zip@
Edit	Copy	Delete	hulk	2d6ccd34ad7af363159ed4bbe18c0e43c681f606877d9ffc96...	hulk@
Edit	Copy	Delete	Lastin	f6e0a1e2ac41945a9aa7ff8a8aaa0cebc12a3bcc981a929ad5...	sssss
Edit	Copy	Delete	watcher	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a...	aaaa

Check All *With selected:* Change Delete Export

Number of rows: 25

Query results operations

Security Design Overview

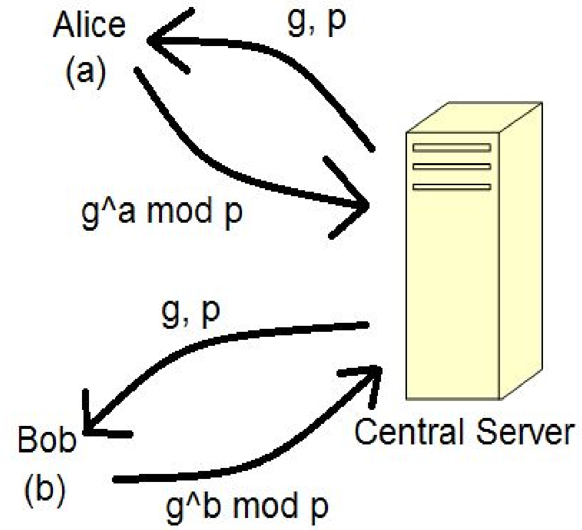
- Initializing the shared key
  - central server to store public keys and user info
  - Diffie-Hellman key agreement to generate a shared key
- Message Encryption
  - symmetric encryption using AES-128
  - compute MAC tag for verification using HMAC SHA-256
- Message Decryption
  - check MAC tag for integrity
  - decrypt with symmetric key

Central Server: Database

- Database
    - stores information about user accounts
        - username/password, email
        - friends list, settings, language preferences, etc.
    - does NOT store chat logs or files
- Security
    - prepared statement to prevent SQL injection
    - password hashing using SHA-256 [9]

Central Server: Diffie-Hellman

- CS chooses some large prime p anc generator g to be the public key pair (p, g). [1, 2]
- User client chooses s between 1 and p-1, which remains secret. [1]
- User client computes g^s (mod p), and sends result to server.
- CS stores list of public keys g^a, g^b, g^c, …for Alice, Bob, Carol, ….



Diffie-Hellman Key Agreement

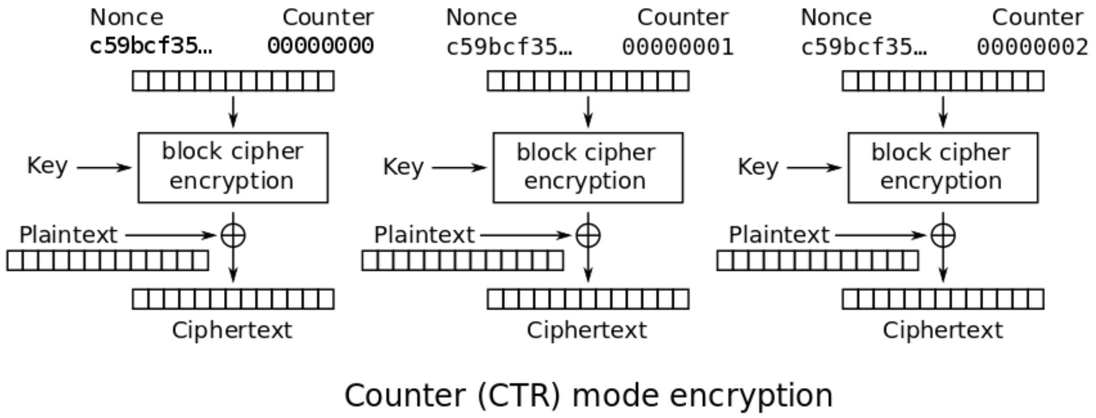
- Alice and Bob want to communicate over CryptalkFUN
    - Alice's client goes to central server and retrieves Bob's public key, g^b, and using Alice's secret a, computes g^(ba) (mod p)
    - Bob's client retrieves Alice's public key, g^a, and using Bob's secret b, computes g^(ab) (mod p)
    - Alice and Bob's shared secret is **g^(ab)**
- Security (Appendix B)
    - **assumption**: central server is secure
    - **justification**: server is acting as central authority [2]
    - no messages are exchanged via Diffie-Hellman

Message Encryption: AES-128

- Alice wants to send Bob a message m
    - message is converted to byte array
    - byte array is encrypted using AES-128
        - CTR (counter) mode: each block is encrypted using the symmetric key and some shared counter [3]
        - CTR mode does not require padding
        - symmetric key K is shared secret from Diffie-Hellman stage
- Alice now has ciphertext c = ENC(K, m)

Message Encryption: AES-128

- CTR Mode:



Source: [10]

Message Encryption: AES-128

- Security (Appendix C, part 2)
    - CTR mode
        - counter changes for each block encrypted, which ensures that the ciphertext of two identical messages will be different [5]
        - counter does not need to be secret: as long as no values are repeated, above property holds [6]
        - does not leak plaintext data patterns as in ECB
    - encryption protects confidentiality
        - message m cannot be read without symmetric key
        - replay attacks will fail, as decryption of same ciphertexts at different counter values will produce different plaintexts

Message Verification: HMAC

- Compute MAC (Message Authentication Code)
  - HMAC: Hash-Based Message Authentication Code [8]

    $$HMAC(K, m) = H\Big((K \oplus opad) \;||\; H\big((K \oplus ipad) \;||\; m\big)\Big)$$

    - hash function H: SHA-256 [9]
    - uses symmetric secret key K
    - changing initialization vector [7]
  - Alice computes a tag for the ciphertext c, tag(c) = HMAC(K, c)
- Alice sends (c, tag(c)) to Bob

Message Verification: HMAC

- Security (Appendix C, part 1)
    - message integrity can be checked by verifying tag
    - message is authenticated if tag is successfully verified, since only two parties have the symmetric key K
    - replay attacks will fail, as changing IV for HMAC means tag(c1) != tag(c2) even if c1=c2, so verification algorithm will reject a replay

Message Decryption

- Bob receives (c, tag(c)) from Alice
  - first verify the tag, and if the verification fails, don't bother to decrypt the ciphertext: the message was tampered with, or it came from the wrong source
  - if verification succeeds, decrypt c and recover message m
- Security (Appendix D)
  - Bob now knows that m comes from Alice and no one else has tampered with the message.

Resources

Java

- cryptography library: javax.crypto package
- provider (for implementation): https://www.bouncycastle.org/latest_releases.html

References

1. Implementaion of Diffie Hellman key generation: https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#KeyAgreement
2. Diffie-Hellman algorithm as defined in PKCS #3: ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-3.asc
3. AES implementation: https://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html
4. Message padding as described in PKCS #5: https://tools.ietf.org/html/rfc2898
5. Security of CTR mode: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf
6. Choice of initial counter vector for CTR: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
7. Implementation of HMAC: https://docs.oracle.com/javase/7/docs/api/javax/crypto/Mac.html
8. HMAC algorithm: http://web.cs.ucdavis.edu/~rogaway/papers/modes.pdf
9. SHA-256 algorithm: https://www.ietf.org/rfc/rfc2104.txt
10. CTR mode image source: https://upload.wikimedia.org/wikipedia/commons/thumb/4/4d/CTR_encryption_2.svg/902px-CTR_encryption_2.svg.png

CryptalkFUN: Secure Two-Way Communication

Che Wang, Liming Luo

Appendices from the presentation slides can be found here.

Appendix A: Motivation for secure two-way communication

1. examples
   a. two spies want to securely exchange information
   b. two students are working on a project together, and they want to make sure that no one else steals their ideas
2. attacks and consequences
   a. eavesdropping
      i. Sensitive messages may be overheard by third parties.
   b. data modification
      i. A message may be intercepted and changed to have a completely different meaning than what was sent.
      ii. Even if an attacker does not know how to intentionally change the meaning of a message, she can affect the functionality of the program with data modification, such as by altering some ciphertext that has been sent such that the decryption makes no sense, or by deleting messages.
   c. replay attacks
      i. May be used to forge authentication, such as by intercepting a valid authentication and then providing it the next time authentication is required. Then the attacker can pose as a legitimate party and obtain sensitive information.
      ii. May be used to send (or resend) a valid message at the wrong time. (For example, sending “goodbye” in the middle of a conversation to interrupt it, thus affecting functionality.)

Appendix B: Diffie-Hellman Key Agreement

1. Central Server

   a. In the Diffie-Hellman algorithm, it is assumed that some trusted central authority will generate the pair (p, g); in our program the central server takes this role. [2]
   b. We make the further assumption that the central server's list of public keys is secure, which we consider reasonable since it is already trusted to provide a correct (p, g) pair to clients. This protects us against man-in-the-middle attacks since either party can go to the server to check that the other party's public key is indeed correct.
   c. If this further assumption does not hold, authentication can be added by using STS protocol between clients directly at this stage.
   d. For security, the central server should periodically do a key refresh—choose a different p, g, and have clients update their secret and public keys. (Appendix C, 1.b.ii is related.) However this was not implemented in our program as it is beyond the scope of our demo.
2. Using the same (p, g) pair for multiple clients results in public keys $g^a, g^b, g^c, g^d, \ldots$. This may pose a security risk if any of $a, b, c, d, \ldots$ are small multiples of each other. (For example, if Alice sees that $g^b = (g^a)^2$, then she knows that Bob's secret is a*2.) However, we assume that p is chosen sufficiently large that this is not likely; if the secrets are sufficiently large multiples of each other, we are once again reduced to solving the discrete log problem.
3. Because no messages are actually exchanged via Diffie-Hellman, our only concern is the secrecy of the shared key, and we do not consider any of the eavesdropping, modification, or replay attacks on data.
4. In our program we choose a key length of 512 bits as implemented by the cryptography library function.
5. The size of the prime p chosen will depend on the level of security required (ex: 512 bits will protect you from your classmate eavesdropping, but probably not the NSA), and choosing a large enough prime is necessary to ensure the secrecy of the symmetric key that will be used in the next step. A discussion of advanced techniques for attacking the discrete log problem, such as index calculus or the number field sieve, is beyond the scope of this project, but we want to choose p large enough to make even these attacks impractical. Alternatively, using Diffie-Hellman on elliptic curves will render these

attacks unusable, so switching to elliptic curves is another solution to ensure secrecy, although this option is not available in the current implementation.

Appendix C: AES-128 and HMAC SHA-256

1. AES-128
    a. We chose AES-128 as our symmetric encryption scheme, because it is currently considered sufficiently secure (for example by the U.S. government, which is always trustworthy, for encryption of top secret data) and it has some encryption speed improvement over AES-192 or AES-256. If, in the future, AES-128 is no longer secure, this stage can be modified to use a longer key length.
    b. We take as the AES symmetric key the first *n* bits of the shared secret g^{ab}, where *n* is the key length of the encryption scheme we are using (in this case, *n* = 128). Since the shared secret is 512 bits, it is sufficient.
    c. CTR mode
        i. The counter value does not need to be secret, as long as no value is used more than once with the same key. Therefore to set (or reset) the counter, Alice and Bob can request some random value from the central server, or use the timestamp, or similar, as a nonce. [6]
        ii. Because each block size is a fixed length of 128, if more than 2^{128} blocks are encrypted with the same symmetric key, then we are guaranteed to have counter values repeat; if even more than 2^{64} blocks are encrypted, there is a high chance that some randomly generated nonce will have a collision with a previously used counter value (according to the birthday problem). However it is very unlikely that anything near 2^{64} blocks will be encrypted before a key refresh occurs.
        iii. Because the encryption of the next block does not depend on the outcome of the encryption of the current block, CTR is easily parallelizable for faster computation (and also allows random read access). This can make encryption of large files more efficient, although sending anything other than text is not supported in the current implementation.

        iv. CTR mode does not require padding as it essentially acts as a stream cipher. Therefore the length of the plaintext is revealed by the length of the ciphertext. However, we do not consider this a major concern; in particular CTR mode is not vulnerable to the padding oracle attack.

2. HMAC SHA-256
    a. Message integrity is checked by verifying the tag as tampering with either c or tag(c) will cause the verification algorithm to fail when Bob receives (c, tag(c)). Any attempts at data modification will be detected and rejected by verification.0
    b. Message authentication is checked by verifying the tag as only two people share the symmetric key, and if the receiver verifies the message and knows that he did not send it to himself, then it must have come from the other party.
    c. If the verification algorithm returns false, the receiver does not know whether the message has been tampered with, it came from the wrong source, or both. In either case the received message is not considered trustworthy and is discarded without decryption.
    d. The HMAC implementation in the java cryptography library has the option for a changing IV, which we use to protect against replay attacks and tag forging. Without the IV, if an eavesdropper intercepts some (c, tag(c)), she may later send the same (c, tag(c)) to Bob, and Bob's verification algorithm will return true. Because the counter in the AES section has incremented, Bob's decryption of the message c will not reveal the original plaintext, but Bob will still think that it was Alice who sent him the message and not the eavesdropper (therefore forging authentication). With a changing IV, this replay attack is not possible as the verification algorithm will compute a different tag(c) for each ciphertext it receives, even if c is the same, and thus reject the replay.

Appendix D: Security Properties

1. Confidentiality: This property is preserved by encryption which makes it difficult for third parties to uncover messages. Any manipulation of the ciphertext will cause the authentication step to fail, and no information about the plaintext will be released.
2. Integrity: This property can be checked by the MAC tag.

3. Authentication: This property can be checked by the MAC tag.
4. Deniability: This encryption scheme ensures that messages are deniable (that is, does not have the property of non-repudiation), which preserves the property of secure two-way communication. If Alice and Bob are communicating with each other, they are the only ones who can decrypt the messages between them, and neither party can prove to anyone else that the other person sent a certain message (i.e, Bob may claim that Alice sent a message, but he may have forged it himself). Therefore, only Bob and Alice—and no third parties—can be sure that a certain chat log is genuine.
5. Perfect Forward Secrecy: Our system does not have the property of perfect forward secrecy; if a secret key is leaked in some future, any ciphertexts encrypted with that key will be decryptable. To preserve forward secrecy the parties would have to use ephemeral keys so that the compromise of one ciphertext will not affect other ciphertexts. However, the drawback to implementing this is that it requires both parties to be online to negotiate keys, which we did not feel was something we wanted to require in this application, particularly as in the future the program may be expanded to an email system.