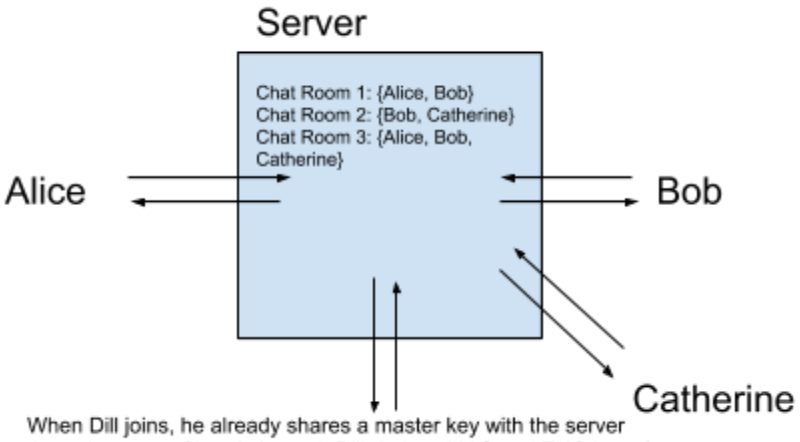
Secura: A lightweight, high-security messaging service

Prikshet Sharma - Acquincum Institute of Technology

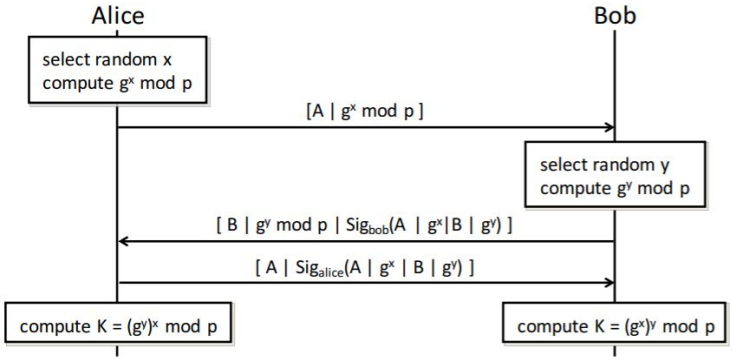


Overview

- Everyone establishes a shared secret with the server using an out-of-band channel.
- For sending encrypted messages, people share keys using the EC ElGamal key exchange protocol.
- Once EC ElGamal is complete for each user in both directions, use these keys to derive new keys using HMAC with SHA-256. New keys are derived for each person after every ten messages in the chatroom. Old keys are delete immediately.
- The server cannot decrypt any message it forwards from one person to another because it is encrypted with keys derived from a EC ElGamal exchange between those people. Therefore, the messages are end-to-end encrypted.
- People send encrypted requests to the server to create new chat rooms, or to join an existing chatroom.
- Signatures are used for user authentication.
- Pseudo Random Number Generator Fortuna is used, which accumulates entropy from many sources, rather than from just boot time.

Attacker Models and Proposed Solutions

- Meet in the Middle Attack and Padding Oracle: Attacker somehow knows message m* that will be sent. Attacker stores $(m*, MAC(m*))$ for $2^{k/2}$ random keys. It intercepts the message and and compares its MAC will his pre-computed MACs. Eventually, after approximately $2^{k/2}$ sessions, the correct MAC key will be found. The attacker cannot guess m*, because it is encrypted and could be any string. All chosen ciphertext attacks, like the Padding oracle, will also fail.
- Replay attacks: Use explicit sequence numbering to prevent replay attacks. Message sequence numbers are sent in the header. We use a receiving window because messages might arrive out of order.
- For semantic security and prevention from adaptive chosen ciphertext attacks, use probabilistic encryption inherent in ElGamal.
- EC ElGamal as our key feature. ElGamal provides semantic security and Elliptic Cryptography makes our messaging service lightweight, due to less space required to store keys.

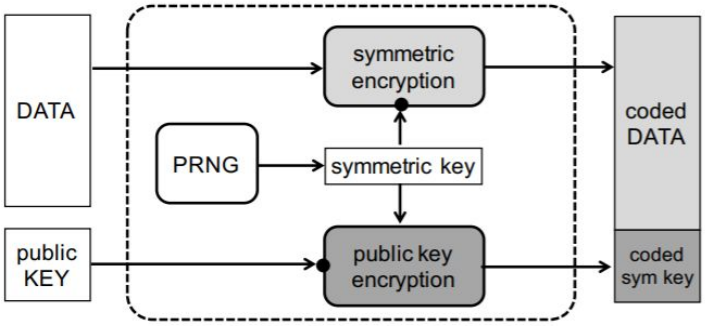


- Key freshness is inherent with ElGamal key exchange. Key authentication is provided with digital signatures as shown in the figure (the figure actually shows Diffie Hellman but the concept is the same):

System Description

- TLS message format
- The user has a sending and receiving sequence number starting from 0 for each chat room he is part of.
- Send_sequence_number is incremented each time a message is successfully sent.
- Encrypt messages using keys derived from keys shared using two-way EC ElGamal key exchange between each pair of people.
- Use hybrid encryption. Encrypt message with a symmetric cryptosystem and encrypt its key with the key derived using EC ElGamal as shown in the figure:
- Header contains sender id and chat room number. Server forwards this message to all people in the chat room. The receiver uses his private key to decrypt the symmetric key and decrypt the message with the symmetric key.

- Server keeps track of the user ids associated with chat room.



EC ElGamal implementation:

The protocol:

Given elliptic curve E in $\{0, , p-1\}$ for a large prime p,

Bob: choose point $A = (x, y)$ on E such that x, y are integers mod p
a number a
point $B = a * A$
public key: (A, B),
private key: a

Alice: (encryption) choose random integer k

$$C1 = k * A$$
$$C2 = m + kB$$

Send $(C1, C2)$ to Bob

Bob: compute $C2 - aC1 = m$ to read the message.

Using **Crypto.PublicKey.ECC**, we can generate an ECC key and store it externally.

Encryption Pseudocode:

$enc(ECCkeykey, msgm)$

M<-$Point(m, key.E)$
E<-$key.curve$
K<-$random\_integer()$
$C1$<-$ECC\_mult(K, key.A, key.E)$
$C2$<-$ECC\_add(M, ECC\_mult(K, key.B, E), key.E)$

$return(C1, C2)$.

Where $PointECC\_mult(scalar, Point, Curve)$ and $PointECC\_add(Point, Point, Curve)$ functions will be implemented following the definition of point addition on ECs.

Decryption Pseudocode:

$dec(ECCkeykey, (C1, C2))$

a<-$key.privateKey$
a<-$(-a)$
$aC1$<-$ECC\_mult(a, C1, key.E)$
M<-$ECC\_add(C2, C1, E)$
$return\ M.x + M.y$

Further Explorations:

- How to represent message m as a Point M on the EC?

Suggested solution:

1. Split m into blocks of size n (add padding as needed)
2. Choose a non-negative number $q <= min(Mi, p - 1)$
3. If $(q, Mi - q)$ is on the EC, represent Mi as $(q, Mi - q)$ and terminate
4. Else repeat with a different q

Note: if $n <= log(p - 1)$, than a block of size n cannot represent a number greater than $(p - 1)$ ($Mi <= p - 1$). Thus, for any $q <= (p - 1)$ it will hold that $|Mi - q| <= (p - 1)$. Thus, point $(a, Mi - q)$ will be on the elliptic curve. The fact that such q is not unique is not a problem, since all we are interested in is in recovering Mi later on, which will be independent from the value of q.

A problem will be if none of the points on the EC 'sum' to Mi for some value of i.

- Base components of ECCkey object are a string describing curve E ("P-256", "prime256v1" or "secp256r1".), a private key q and a public key qPoint.
    - How to work with curve E arithmetically, given an ECCkey?
    - How to extract the prime p, which may be needed for splitting the message into blocks?
    - How to extract the Point such that $q * Point = qPoint$, which is needed for implementation of EC Elgamal?
- Since EC Elgamal public key is a pair of points, what is the best way to use the ECCkey object?