Lab 2: Exploring Public Key Cryptography

Report by Ryan Blair, 9:30am section

Task I. Implement Diffie Hellman Key Exchange

How hard would it be for an adversary to solve the Diffie Hellman Problem (DHP) given these parameters? What strategy might the adversary take?

A: The problem would take an exponential amount of time because the attacker has to determine the secret key using the $\log_g(g^a)$ problem. With a small generator, however, this problem doesn't take too long to brute force it. If necessary, an attacker can just use a MITM attack where they intercept the passed value(s) (A, B) between the two parties, encode their own shared value using the large prime they share, and then pass that one to Alice (who still believes she is communicating with Bob) or Bob.

Now, let's use some "real life" numbers. Modify your implementation to use these parameters, make sure Alice and Bob can compute the same symmetric key k, and correctly exchange some encrypted messages.

Would the same strategy used for the tiny parameters work here? Why or why not?

A: Yes, because the only real change that has happened is the length of our public prime number and generator, so the same strategy will work.

Task II. Implement MITM Key Fixing & Negotiated Groups

A. Modify your implementation from Task I in the following way:

In your implementation, show that Mallory can successfully decrypt the messages c_0 and c_1.

(A = p; B = p)
sharedA = pow(p, a, p) = 0
sharedB = pow(p, b, p) = 0
kA = b'_\xec\xebf\xff\xc808\xd9Rxlmily'
kB = b'_\xec\xebf\xff\xc808\xd9Rxlmily'
m0 = Hi Alice!
m1 = Hi Bob!

B. Repeat this attack, but instead of tampering with A and B, tamper with the generator g. Show that Mallory can recover some of the messages by setting g to 1, p, or p-1.

(g = 1)
A = pow(g, a, p) = 1
B = pow(g, b, p) = 1
sharedA = 1
sharedB = 1

kA = b'k\x86\xb2s\xff4\xfc\xe1\x9dk\x80N\xffZ?W'
kB = b'k\x86\xb2s\xff4\xfc\xe1\x9dk\x80N\xffZ?W'

(g = p)
A = pow(g, a, p) = 0
B = pow(g, b, p) = 0
sharedA = 0
sharedB = 0
kA = b'_\xec\xebf\xff\xc808\xd9Rxlmily'
kB = b'_\xec\xebf\xff\xc808\xd9Rxlmily'

(g = p-1)
A = 1243253391...20912
B = 1243253391...20912
sharedA = 1243253391...20912
sharedB = 1243253391...20912
kA = b'\xce\xbd\x8b\x1a\xe1Y\x1f\xc4\xe1\x86[ClK\xd5\x12'
kB = b'\xce\xbd\x8b\x1a\xe1Y\x1f\xc4\xe1\x86[ClK\xd5\x12'

Why were these attacks possible? What is necessary to prevent them?
A: They were possible because the shared values from Alice and Bob were the same, and it's something an attacker can calculate. To prevent this from happening, the key needs to incorporate a signature that both parties can tell when a message has been tampered.

Task III. Implement "textbook" RSA & MITM Key Fixing via Malleability

A. Encrypt and decrypt a few messages to yourself to make sure it works.

While it's very common for many people to share an e (common values are 3,7, 2^{16}+1), it is very bad if two people share an RSA modulus n*. Briefly describe why this is, and what the ramifications are.*
A: If people shared the same modulus n, then, once one key has been compromised, others who share that modulus would be compromised as well.

B. Find the operation F(·) that Mallory needs to apply to the ciphertext c that will allow her to decrypt the ciphertext c_0. HINT: Mallory knows Alice's public key (n,e) and can encrypt her own messages to Alice.
Code:
cip = rsaEncrypt(msg, public)
cip = 0
mal = rsaEncrypt(msg_m, public)
dec = rsaDecrypt(mal, secret)

Give another example of how RSA's malleability could be used to exploit a system (e.g. to cause confusion, disruption, or violate integrity).
A: Since an attacker already has the user's public key, Mallory could make their own ciphertext using said public key and disrupt the system, where it doesn't know whether the user or the impersonator is authentic (assuming no signatures have been established).

Malleability can also affect signature schemes based on RSA. Consider the scheme:

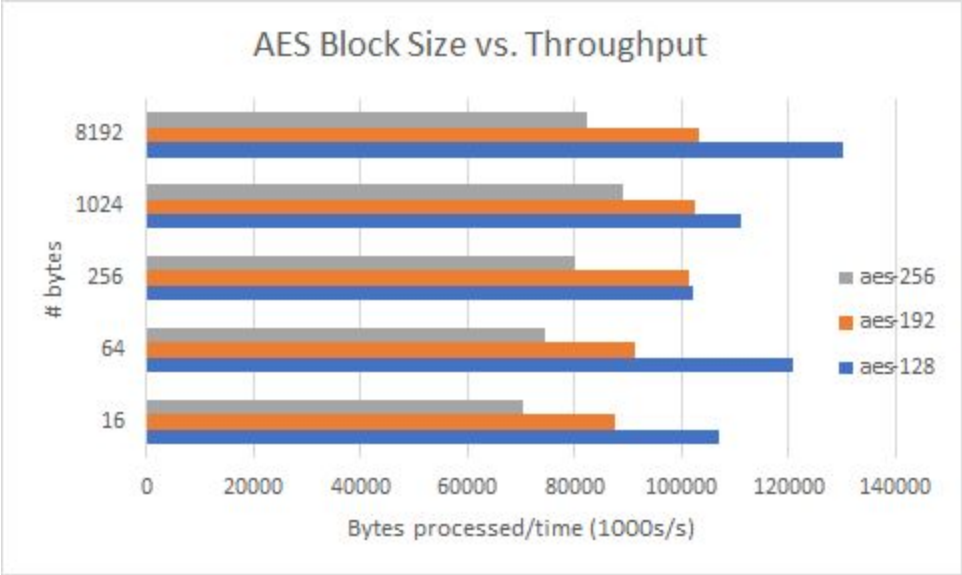
$Sign(m,d) = m^d \bmod n$

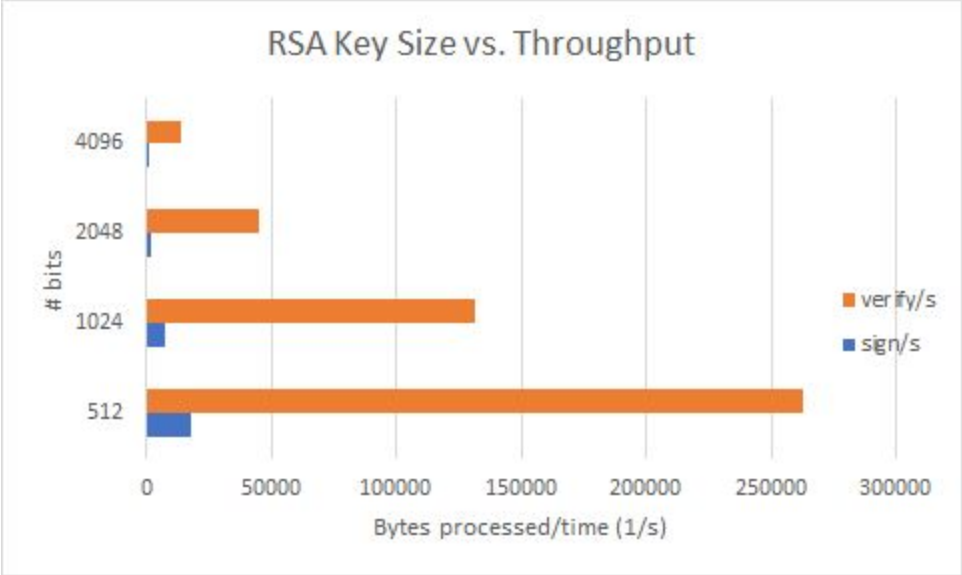
Suppose Mallory sees the signatures for two message m_1 and m_2. Show how Mallory can create a valid signature for a third message, $m_3 = m_1 \cdot m_2$.
A: $Sign(m_3, d) = Sign(m_1, d) * Sign(m_2, d)$
$m_3^d \bmod n = m_1^d \bmod n * m_2^d \bmod n$
$m_3^d \bmod n = (m_1 * m_2)^d \bmod n$

C.
Briefly justify whether either of the following key exchange protocols do or do not provide forward secrecy.
A: According to Wikipedia, Diffie-Hellman exhibits forward secrecy because it generates new key pairs each session and discards them after. This is because its' fast key generation so even if an eavesdropper figures out one key, that does not mean it will know future keys.
RSA, on the other hand, does not provide forward secrecy because someone can impersonate another just through their public key. Each public key is assigned, so once it's known, then it won't be changed until the next update.

Task V.
`openssl speed rsa`
and
`openssl speed aes`

Run these operations and report your findings. Include two graphs: one that plots the block size vs. throughput for the various AES modes of operations and one that plots the RSA key size vs. throughput for each RSA operation. How do the results compare?





A: The performance of RSA encryption exponentially falls off as the size of the key increases. AES encryption, on the other hand, becomes overall more effective as the key size increases. RSA takes a long time to sign keys compared to simply verifying them. It's also noteworthy that AES-128 has a better performance than AES-256 and AES-192 because the key lengths are larger as it takes longer to encrypt the keys.