

Information Assurance: Homework 5 – Answer key

Due October 13, 2006.

1. The strength of the RSA algorithm is based on the difficulty of factoring large prime numbers. Assume you are given the factorization for the modulus of a public key ($n = p \times q$). Show now this breaks the RSA key pair.

If we know $n = p \times q$, we can easily compute $\Phi(n) = (p-1)(q-1)$. The public key exponent e is relatively prime to n and the private key exponent d is the inverse of e modulo $\Phi(n)$, i.e., $e \cdot d \bmod \Phi(n) = 1$. If we know the totient and e , we can compute d . Once we know d the key pair is broken.

2. Suppose Alice and Bob have RSA public keys in a file on a server. They communicate regularly using authenticated, confidential messages. Eve wants to read the messages but is unable to crack the RSA private keys of Alice and Bob. However, she is able to break into the server and alter the file containing Alice's and Bob's public keys.
 - a. How should Eve alter that file so that she can read confidential messages sent between Alice and Bob, and forge messages from either?

Assume Alice has a public key (e_A, n_A) and Bob has a public key (e_B, n_B) registered. Eve could create two new key pairs (e_{EA}, n_{EA}) and (e_{EB}, n_{EB}) and register them for Alice and Bob. Then if Eve can intercept all communication exchanged between Alice and Bob, she can undo their computation and replace it with a signature/encryption using her version of the key. E.g., suppose Bob sends a signed encrypted message to Alice $\{m \{h(m)\}_{d_B}\}_{e_A}$. Eve would intercept it, decrypt it using her version of Alice's private key, recalculate the hash using her version of Bob's private key, and re-encrypt using her version of Alice's public key to compute $\{m \{h(m)\}_{d_{EB}}\}_{e_A}$.

- b. How might Alice and/or Bob detect Eve's subversion of the public keys?

If Eve fails to intercept a message the deception is found, because the messages destined to Alice and Bob will not correctly decrypt, since Alice and Bob will use their real private keys to decrypt rather than Eve's fake private keys.

Alternatively, Alice and Bob should periodically verify that the public key they have posted on the key server is correct. They could also post a fingerprint of their public key at a separate location (e.g. On their personal web site or at the end of emails) to encourage people they communicate to validate what they receive from the certificate server.

3. Thanks to the birthday paradox one can find collisions using the DES-MAC cryptographic hash function in 2^{32} messages. Alice wants to take advantage of that fact to make it swindle Bob. She has two contracts. One that Bob is willing to sign

and another that Bob is not willing to sign. She needs to generate a version of each that has the same DES-MAC crypto hash. Suggest how she might do this. Hint: adding white space and combinations of characters with back spaces do not change the meaning of the contracts.

Using DES-MAC with 64 bits, there are 2^{64} possible hashes. Just using the pigeon hole principle, you are very likely to find a match for a particular hash after trying 2^{64} different messages. But in this case, you can vary both the original contract $M1$ and your goal contract $M2$. In this case, you have the birthday paradox. You are not trying to match a particular hash value, rather given two groups, you want to find one member in each group that matches each other. If you compute hashes of 2^{32} versions of $M1$ and 2^{32} versions of $M2$, then you have a probability of around $\frac{1}{2}$ that you have a match between the variants of $M1$ and the variants of $m2$.

4. Work with Gnu Privacy Guard (GPG) or Pretty Good Privacy (PGP). They both implement the same protocols, but PGP uses proprietary encryption algorithms. You can access free trial versions of PGP from <http://pgp.com>. I have used the Windows version. You can access GPG from <http://gnupg.org>. I have used this on Linux and installed it via yum on my personal system. It may already be installed on the University Linux systems. Type “man gpg” to see. I will be evaluating your results on my Linux box using GPG. So if you use PGP, be sure to create a key using some combination of DSA and ElGamel (algorithms supported by GPG). Once you get your GPG/PGP system operational perform the following tasks:
 - a. Create a key pair.
 - b. Get your key signed by at least one other person. Submit an exported version of your signed public key.
 - c. Encrypt a file using the instructor’s public key (at <http://www.cs.uiuc.edu/class/fa06/cs498sh/hw5/skh-pubkey.asc> with fingerprint 388E 7466 4DD3 390E 8F36 A535 474D 5DC9 4912 BF7E) and sign it with your key. Submit the signed and encrypted file.

This one was pretty straight forward. Some people didn't submit a signed key and lost a few points on that. One thing I noticed is for a fair number of submissions I had to decrypt the encrypted file twice. I'm not sure why that would be the case. It could be that the first decrypt was just verifying a signature. The first decrypt was definitely not using my key. I would only be prompted for my pass phrase (thus accessing my private key) on the second decrypt.

5. In the Otway-Rees protocol, both a session id (n) and nonces ($rand_1$ and $rand_2$) are used. Are both really needed? Would the protocol be equally resilient if only the session id or the nonces were used? Explain why or why not.

Yes, both the session ID and the nonces are needed. Assume that we only had the session id. One problem is the first two message exchanges, we would be handing Eve a pair of known ciphertext/plaintext pairs which could potentially weaken k_{Alice} and k_{Bob} .

In messages 3 and 4, this is nothing to prevent Eve from replacing one or both of the encrypted session keys Cathy's message with previously generate sessions that Eve has presumably broken. Eve could simply pre-pend the encrypted keys with the currently used session ID.

Assume we only have the nonces and not the session ID. The session ID was giving us a means to track that a particular response matches a particular request. The nonces almost give us that. Alice and Bob could note that R1 or R2 is in the current request and reject responses that contain a different nonce. This has two issues.

- 1) Random numbers may repeat, so either Alice and Bob must remember all previous nonces (or all nonces within a window) to prevent Eve from getting luck.*
- 2) With a common tracking number, Carol can verify that Alice's and Bob's requests really belong together.*

These both seem like rather obscure attacks, but as a number of you noted, adding another session ID is pretty check and simplifies the implementation of the session tracking.

One solution to the replay problem in the session ID only scenario is that Cathy could include the session ID in the encryption with the session key. $num || \{ k_{session} || num \}_{k_{Alice}}$ $|| \{ k_{session} || num \}_{k_{Bob}}$. This gives us some of the replay protection that nonces give us, but in this case, Bob has no choice in the replay value. Eve could replay the first message to Bob to make him select a session ID of Eve's choosing. The nonce solution lets each party pick their own replay value and avoids this problem. So you could have two session ID's, one selected by each party. You still have the problem of handing Eve known plaintext/ciphertext pairs in the first two messages, so you would still probably want to introduce secret random values in the encrypted portions of the first two messages.