650.445/600.454: Practical Cryptographic Systems September 19, 2018

Written HW 2

Instructor: Matthew Green *Due: 11:59 pm September 27, 2018*

Name: _____

The assignment must be completed individually. You are permitted to use the Internet and any printed references. In particular, you may find the Handbook of Applied cryptography (available online) useful for several of these questions.

Please submit the completed assignment via Blackboard.

Problem 1: Short answers. (You don't need to justify your answer, but you might want to for partial credit.)

1. Consider the following variant of the Encrypt-and-MAC construction:

$$C = \mathsf{Encrypt}(k_1, M) \| \mathsf{MAC}(k_2, M \| \mathsf{ctr})$$

   Where ctr is a counter that increments after each message is encrypted, such that the same value is never used twice. Explain why this *might* not be insecure.

2. Describe the consequences (on the plaintext, following decryption) of flipping a single bit in a CFB mode ciphertext.

3. Explain why it might not be safe to use a small public RSA exponent (for example, $e = 3$) in some cases.

4. Why is it unsafe to share the same prime between two different RSA public moduli (*e.g.,* imagine I generate $N_1 = pq$ and $N_2 = p'q$ where $p \neq p'$ and publish N_1, N_2 as public keys)?

5. The Imperfect Diffie-Hellman paper mentions that solving the discrete logarithm on 512 bit Diffie-Hellman keys can take a week of computation. What made it possible to solve discrete logarithms quickly enough that a solution could be found within the lifetime of a TLS or IKE handshake (typically minutes, at most)?

Problem 2: Longer answers

1. **Symmetric key distribution.** Review the Needham-Schroeder shared-key key distribution protocol shown in Figure 1.[1] This protocol allows two parties (A and B) to agree on a shared key, using a trusted party T (with whom each party already

[1] This diagram is excerpted from §12.26 of the HAC. It should not be confused for the Needham-Schroder *public key* protocol, which is a different protocol entirely!

Protocol Needham-Schroeder shared-key protocol

SUMMARY: A interacts with trusted server T and party B.
RESULT: entity authentication (A with B); key establishment with key confirmation.

1. *Notation.* E is a symmetric encryption algorithm (see Remark 12.19).
   N_A and N_B are nonces chosen by A and B, respectively.
   k is a session key chosen by the trusted server T for A and B to share.
2. *One-time setup.* A and T share a symmetric key K_{AT}; B and T share K_{BT}.
3. *Protocol messages.*

$$\begin{array}{lll} A \to T: & A, B, N_A & (1) \\ A \leftarrow T: & E_{K_{AT}}(N_A, B, k, E_{K_{BT}}(k, A)) & (2) \\ A \to B: & E_{K_{BT}}(k, A) & (3) \\ A \leftarrow B: & E_k(N_B) & (4) \\ A \to B: & E_k(N_B - 1) & (5) \end{array}$$

4. *Protocol actions.* Aside from verification of nonces, actions are essentially analogous to those in Kerberos (Protocol 12.24), and are not detailed here.

Figure 1: Needham-Schroeder protocol, excerpted from §12.26 of the Handbook of Applied Cryptography. The terms "A" and "B" refer to the *identities* of the parties. The *nonces* N_A, N_B (literally, "number used once") are fresh and hopefully random strings generated by A and B respectively.

shares a key) as an introduction point. You should assume that A, B and T are all trustworthy, and that the encryption scheme provides strong confidentiality. You can also assume that the attacker controls the network and is allowed to eavesdrop, block or *modify* any messages sent between the parties.

Answer the following questions:

(a) Assuming no fancy attacks on the encryption scheme, what prevents an evil user (C) from impersonating A to B?

(b) What are the nonces N_A, N_B needed for?

(c) Why does the final message encrypt $E_k(N_B - 1)$ rather than $E_k(N_B)$? What attack might be possible if the subtraction was removed, and the final message simply contained $E_k(N_B)$?

(d) In the second (and third) message, why does the ciphertext $E_{K_{BT}}(k, A)$ encrypt the identity A?

(e) Imagine that the encryption scheme E is implemented using AES-CTR with no MAC. What attacks could you come up with against this protocol?

(f) What happens to this protocol if A is ever compromised and all of its data (keys etc.) are stolen?

2. **Diffie-Hellman and RSA**. Answer the following questions:

   (a) List all of the *subgroups*[2] of $\mathbb{Z}_{23}^*$ and provide one generator for each. See HAC §2.5 for definitions if this is helpful.

   (b) What is the order of the group $\mathbb{Z}_{23}^*$. List the prime factors of the order of the group.

   (c) Describe a simple algorithm for solving the discrete logarithm of a value $h \in \mathbb{G}$ base g, that is: finding an x such that $g^x \equiv h \ mod \ p$. Your algorithm can be inefficient.

   (d) Let p be a prime and let g be a generator of the group $\mathbb{Z}_p^*$, where (p, g) are used for Diffie-Hellman. Imagine that Bob re-uses the same secret key b for many Diffie-Hellman connections. How can Alice learn (at least) one bit of b? Hint: think about how you could do it in your toy $p = 23$ group above.

   (e) A trusted central party wants to make and distribute many RSA keypairs that all share a single public modulus $N = pq$. For each of n users in the system she generates a different public/secret exponent pair $(e_1, d_1), (e_2, d_2), \ldots, (e_n, d_n)$ that work with N, and sends each exponent pair to a party so that the i^{th} party's public key is (N, e_i). What is the risk of this system?

[2]Note: a *subgroup* is a cyclic group that is contained within the larger group. You can find subgroups by doing what we did in class: picking an element of the group and seeing whether it generates the whole group or just a subset.