CS 127/CSCI E-127: Introduction to Cryptography

Problem Set 9

Assigned: Nov. 15, 2013 Due: Nov. 22, 2013 (5:00 PM)

Justify all of your answers. See the syllabus for collaboration and lateness policies. Submit solutions by email to `kevin@seas` (and please put the string "CS127PS9" somewhere in your subject line).

Problem 1. (CBC-MACs)

a) As discussed in KL2e §4.4, the basic CBC-MAC is insecure when the number of blocks in the message is not fixed. As shown in KL2e §4.4.2, this can be fixed by prepending a message m with its length $|m|$ and then computing the CBC-MAC on the result. In contrast, show that appending the message length $|m|$ (as an additional n-bit block) to the *end* of the message m before applying CBC-MAC does not result in a secure variable-length MAC. (Hint: query for the MACs of messages of the form m_1, m_1', and $m_1 \| m_2 \| m_3$, where $m_1, m_1', m_2, m_3 \in \{0,1\}^n$ and m_2 is chosen appropriately to enable a forgery for another message.)

b) Extra credit: In class, we saw a general method for transforming a secure MAC for short messages into a secure MAC for long messages. It is natural to ask whether the CBC construction can be used instead. That is, if $\mathcal{F}_n = \{f_k : \{0,1\}^n \to \{0,1\}^n\}$ is *any* secure MAC for message space $\{0,1\}^n$ (without necessarily being a pseudorandom function family), does it follow that CBC-MAC constructed using $\mathcal{F}_n$ is a secure MAC for message space $\{0,1\}^{\ell \cdot n}$? Show that the answer is no, i.e. under standard cryptographic assumptions, there are secure MACs $\mathcal{F}_n$ for which the resulting CBC-MAC is insecure.

Problem 2. (Textbook Rabin Signatures) In this problem, you will show that the "plain trapdoor function" signature scheme using Rabin's functions in completely insecure against a chosen-message attack.

We have set up a signer that generates signatures for chosen messages $\mathbf{m}$. You can query the server for a signature at

`http://netlab-2.eecs.harvard.edu:5000/oracle/name/m`

where **name** is your last name and $\mathbf{m}$ is the message you want the server to sign. Given this "oracle" access to the signer and any other tools you want (e.g. Wolfram Alpha), determine the private keys p, q. Note that the signer will only sign messages $\mathbf{m} \in \mathrm{QR}_N$. Explain in prose how and why your attack works. (Hint: Recall the proof that Rabin's functions are one-way under the Factoring Assumption.)

Problem 3. (Off-line/On-line Signatures) Public-key signatures are quite expensive. The idea of designing off-line/on-line signatures is to split the signing process into two components. The off-line component prepares some information before the message to be signed is known. This

component can afford to be a bit slow and use a full-fledged signature scheme ($\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}$). The on-line component is performed after the message m arrives. It uses the pre-computed information together with m to produce the final signature. The on-line signature component should be "fast".

For each of the following two proposals for off-line/on-line signatures, describe how the verification algorithm would work and determine (with proof) whether the construction yields a secure (off-line/on-line) signature scheme.

a) In analogy to hybrid encryption, in the off-line phase we select a random key k for a secure MAC, and compute $\sigma_1 = \mathsf{Sign}_{sk}(k)$. In the on-line phase, given message m, we compute and send the signature $(k, \sigma_1, \mathsf{Mac}_k(m))$.

b) Here we use a one-time signature scheme instead of a MAC. In the off-line phase, we generate a random key pair (pk', sk') for the one-time signature and compute $\sigma_1 = \mathsf{Sign}_{sk}(pk')$. In the on-line phase we compute and send the signature $(pk', \sigma_1, \mathsf{Sign}'_{sk'}(m))$.

Problem 4. (Merkle Trees) Read about Merkle Trees in KL2e §5.6.2. These provide an alternative method for domain extension of collision-resistant hash functions, by hashing in a binary tree instead of a linear path.

a) Prove KL2e Theorem 5.11, showing that the Merkle Tree construction yields a secure collision-resistant hash function, if built from an initial collision-resistant hash function.

b) As described in KL2e §5.6.2, Merkle Trees give an efficient solution to the following data integrity problem. A client wishes to upload files $x_1, \ldots, x_t$ to a server, and have a way of checking that when she later downloads any of the files x_i, the server returns it to her completely unmodified. To do this without having to keep the entire files herself, the client could store $H(x_1), \ldots, H(x_t)$ for some collision-resistant hash function H, and then just check the hash of a file she receives from the server. Merkle Trees allow the client to solve the problem with even less space: instead of storing hashes of all the files, the client only has to store the hash value at the root of the Merkle Tree.

   Now, when downloading a single file x_i she also asks for a "certificate" consisting of all the hash values along the path from $H(x_i)$ down to the root, and all the hash values *adjacent* to those along this path (see KL2e §5.6.2). This allows the client to verify all of the hash values leading up to the root, in time and space that grows only logarithmically with t.

   Provide a formal definition of the integrity guarantees provided by this scheme (in terms of a security game played by an adversarial server) and prove that the Merkle Tree scheme meets your definition.