Applied Cryptography

Symmetric Cryptography, Assignment 1, Wednesday, January 31, 2024

Exercises with answers and grading.

1. **(15 points)** In the `PyCryptodome` package, you can find an implementation for `AES-128-ECB`. `AES-128-ECB` applied to a 128-bit input is just the block cipher `AES-128`. Different *modes* can be built using this block cipher, such as `CBC`.

   (a) Implement `AES-128-CBC` and its inverse using the implementation for `AES-128-ECB`. Remember that in CBC mode, the initial value (IV) needs to be random. In particular, your implementation of `AES-128-CBC` needs to take as input a 128-bit value as an IV and a plaintext, which can be of any length. Encrypt a plaintext with an IV and key of your choice, and decrypt the ciphertext again to verify your implementation. The plaintext must be at least 512-bits. Note that the decryption function also takes IV as an input.

   (b) Remember from the lecture that `AES-128-CBC` requires messages of length a multiple of 128 bits. To deal with arbitrary length messages, we have to use a padding function. A simple padding function appends a 1 and a sufficient number of 0s. Another commonly used padding is the *PKCS7* padding. The *PKCS7* padding works on bytes. Given a message M of length an integral number of *bytes*, it completes the message with enough bytes to ensure that the length of the message is a multiple of 16 bytes:

   - If the message needs one byte of padding, then $\text{pad}(M) = M \| \texttt{0x01}$, where `0x01` is in hexadecimal.
   - If the message needs two bytes of padding, then $\text{pad}(M) = M \| \texttt{0x02} \| \texttt{0x02}$, where `0x02` is in hexadecimal.
   - And so on.

   The *PKCS7* padding always appends at least 1 byte and at most 16 bytes. Implement a verification function $\texttt{VFY}_K$ on top of your `AES-128-CBC` implementation, that operates as follows: on input of a 128-bit IV and a ciphertext C of length a multiple of 128 bits, it returns $\top$ if the padding in the decryption is correct and it returns $\bot$ otherwise. Note that you do not have to implement $\texttt{AES-128}^{-1}$; you can use `PyCryptodome`'s implementation as a building block for your implementation of $\texttt{VFY}_K$.

   (c) Assume an attacker has access to $\texttt{VFY}_K$ with q queries. Implement an attack that, given a 32-byte ciphertext $C_1 \| C_2 = \texttt{AES-128-CBC}_K(M_1 \| M_2)$, recovers M_2. **Hint**: Choose a random C_1', does $\texttt{VFY}_K$ reveal any information?

   (d) This attack has been used in the real world to attack SSL 3.0: `https://www.openssl.org/~bodo/ssl-poodle.pdf`. What measures were taken in TLS 1.0 to prevent this attack?

Begin Secret Info:.....

See `implementation_exercise_2.py` for the script we wrote for the exercise.

(a) Implementing `CBC` given the block cipher $\texttt{AES}_K$ is easy: just take a block M_i, XOR it with C_{i-1} and use $C_{i-1} \oplus M_i$ as input to E_K to get C_i. (For C_0 take the IV.)

(b) The implementation of PKCS7 is straightforward. For $\texttt{VFY}_K$, check the last byte of the decrypted message, verify that this last byte appears as many times as its value, and remove these bytes. If it does not pass these checks, return $\bot$.

(c) *Suppose* $M_1'\|M_2' = \texttt{AES-128-CBC}_K^{-1}(C_1'\|C_2)$. *If* $\texttt{VFY}_K(C_1'\|C_2) = \perp$ *then we learn that* $M_2' = \texttt{AES-128-CBC}_K^{-1}(C_2) \oplus C_1$ *is not correctly padded. We change* C_1' *until the padding is accepted, then we know that either the last byte is* 0×01*, the last two bytes are* 0×02 *and so on. We can change* C_1 *to ensure that the ending is* 0×01*. Hence we learn the last byte of* M_2*, we repeat until we find all of* M_2*.*

Now that we know the last byte of M_2*, we choose* C_2 *so that is always sets to* $0x02$ *in the decription, then we query the oracle until we get* $\top$*, from which we learn the second-to-last byte of* M_2*. We iterate this pattern, recovering* M_2 *in an aoround* 16^2 *queries.*

(d) *Use of the flag* `TLS_FALLBACK_SCSV` *prevents downgrade attacks such as before*

End Secret Info .....

2. **(15 points)** In the lecture, we learned that there are two main types of MAC designs: Wegman-Carter (and Wegman-Carter-Shoup) and Protected Hash. Leaving aside key technicalities, it was explained that CBC-MAC follows the Protected Hash paradigm.

   (a) For each of the following MAC functions, perform a brief literature study, and indicate whether they are roughly following Wegman-Carter (or Wegman-Carter-Shoup) or Protected Hash:

   - PMAC: `https://eprint.iacr.org/2001/027.pdf`
   - CMAC: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38B.pdf`
   - EHtM: `https://www.iacr.org/archive/fse2010/61470235/61470235.pdf`
   - EliMAC: `https://tosc.iacr.org/index.php/ToSC/article/download/10979/10412/10267`
   - EWCDM: `https://eprint.iacr.org/2016/525.pdf`

   Briefly explain your answer.

   (b) Wegman-Carter(-Shoup) requires a nonce, which should not be reused for message authentication. Suppose we instantiate the universal hash function using GHASH, so $H_L = \text{GHASH}_L$, and the attacker can repeat evaluations for the same nonce. Explain how the attacker can recover the key L. **Hint**: focus on messages of length one block, i.e., on messages M such that $|M| = |L|$.

   (c) What is the impact on hardware requirements for these two designs, given the advantages and disadvantages? Which design is more intensive? Can any of them be parallelized?

Begin Secret Info:.....

(a) *PMAC, CMAC, EliMAC and EWCDM follow the Protected Hash paradigm, whereas EHtM has a Wegman-Carter-Shoup design.*

(b) *Fix two different message blocks* M, M' *and evaluate* $T = \texttt{GHASH}_L(M) \oplus E_K(N)$ *and* $T' = \texttt{GHASH}_L(M') \oplus E_K(N)$*. As* M, M' *are single blocks, adding these evaluations leads to* $T \oplus T' = L \otimes (M \oplus M')$ *and thus* $L = (T \oplus T')(M \oplus M')^{-1}$*.*

(c) *The parallel aspect of Wegman-Carter-Shoup can be implemented directly in the hardware, which increases the efficiency of the hash function. However, the use of a nonce requires a source of randomness, which is non-trivial! Protected Hash on the other hand does not need a source of randomness in the hardware, but you pay for this in speed.*

End Secret Info .....

3. **(10 points)** This question asks you to show the equation of lecture 2 slide 12 is not **tight**:

$$\mathbf{Adv}^{\text{unf}}_{\mathsf{MAC}}(q_m, q_v) \leq \frac{q_v}{2^t} + \mathbf{Adv}^{\text{prf}}_{\mathsf{MAC}}(q_m + q_v)$$

In other words, you have to investigate a MAC function that is unforgeable but not PRF-secure. To construct such function, suppose we are given a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Consider the MAC function:

$$\mathsf{MAC}_K(M) = F_K(M) \parallel F_K(M)\,.$$

(a) Prove that MAC is unforgeable up to the bound $q_v/2^n$, i.e., that:

$$\mathbf{Adv}^{\text{unf}}_{\mathsf{MAC}}(q_m, q_v) \leq \frac{q_v}{2^n} + \mathbf{Adv}^{\text{prf}}_{F}(q_m + q_v)\,.$$

You do *not* have to *explicitly* write a reduction from the unforgeability of MAC to the PRF-security of F. What is important is that you can show why the $\frac{q_v}{2^n}$ term appears.

(b) For PRF-security, we consider the setup of a distinguisher that has access to either $\mathsf{MAC}_K : M \mapsto T$ or to a random oracle $\mathsf{RO} : M \mapsto T$. Consider the following distinguisher $\mathcal{D}$:

- Fix an arbitrary M and query the oracle on M to receive a tag T;
- If the left and right half of T are equal, return 1. If the left and right half of T are unequal, return 0.

Determine the exact PRF-advantage of this particular distinguisher $\mathcal{D}$, $\mathbf{Adv}^{\text{prf}}_{\mathsf{MAC}}(\mathcal{D})$.

Begin Secret Info: .....

(a) A first step replaces F_K by a random function f. As F_K is evaluated for q_m+q_v different inputs, this step comes at a cost of $\mathbf{Adv}^{\text{prf}}_{F}(q_m + q_v)$. (This step is very comparable to the first step of the security of CTR-mode.)

Now, consider the MAC function $f(M) \parallel f(M)$. Any forgery attempt has a tag of the form $T = X\|Y$. If $X \neq Y$, the forgery succeeds with probability 0. If $X = Y$, the forgery succeeds with probability $1/2^n$. As the adversary can make q_v forgery attempts, its success probability is at most $q_v/2^n$.

(b) The PRF-advantage of $\mathcal{D}$ is defined as

$$\mathbf{Adv}^{\text{prf}}_{\mathsf{MAC}}(\mathcal{D}) = \left|\mathbf{Pr}\left(\mathcal{D}^{\mathsf{MAC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{\mathsf{RO}} = 1\right)\right|\,. \tag{1}$$

We define $\mathcal{D}$ to return 1 iff the left and right half of T are equal. Thus, if $\mathcal{D}$ is conversing with the real world MAC_K, it always outputs 1:

$$\mathbf{Pr}\left(\mathcal{D}^{\mathsf{MAC}_K} = 1\right) = 1\,.$$

On the other hand, if it is conversing with the ideal world RO, it outputs 1 with probability $1/2^n$:

$$\mathbf{Pr}\left(\mathcal{D}^{\mathsf{RO}} = 1\right) = 1/2^n\,.$$

We conclude for (1):

$$\mathbf{Adv}^{\text{prf}}_{\mathsf{MAC}}(\mathcal{D}) = 1 - 1/2^n\,.$$

End Secret Info .....