Problem Set 6 Solutions

Problem 17. *Fix a blockcipher E with an 8-byte (64-bit) blocksize. Consider the following generalization of CBC to allow the encryption of arbitrary byte strings. Given a byte string M, let* $\mathrm{pad}(M)$ *be M followed by enough bytes to take you to the next multiple of eight bytes, where the extra bytes are one of:* 01*, or* 02 02*, or* 03 03 03*, and so on, up to* 08 08 08 08 08 08 08 08 *(all of these constants written in hexadecimal). Let* CBC2 *be the variant of CBC$ encryption that encrypts M by applying CBC, over E, with a uniformly random IV, to* $\mathrm{pad}(M)$*.*

The CBC2 *method is specified in Internet Standard RFC 2040. Note that a* CBC2 *ciphertext for M will have the form $C = IV \,\|\, C'$ where $|IV| = 64$ and $|C'|$ is the least multiple of 64 exceeding $|M|$.*

17.1. *Do you think that* CBC2 *achieves "good" (at least birthday-bound) ind$-security when E is a good PRP? Why or why not?*

Yes, CBC2 is ind$ secure. Let A^f be an adversary that attacks CBC2. We construct another adversary B^g that attacks CBC$ as follows. The adversary B runs A as a black box. For each query X of A, the adversary B will pad the plaintext as indicated by CBC2, and then gives $g(\mathrm{pad}(X))$ to A. Finally, B answers whatever A answers. The adversary B is almost as efficient as A, and $\mathbf{Adv}^{\mathrm{ind\$}}_{\mathrm{CBC\$}}(B) = \mathbf{Adv}^{\mathrm{ind\$}}_{\mathrm{CBC2}}(A)$.

17.2. *Write a careful fragment of pseudocode for an algorithm $\mathcal{D}$ to decrypt a byte string C under* CBC2*. Have $\mathcal{D}(K, C)$ return the distinguished symbol $\perp$ if it is provided an invalid ciphertext; otherwise, it returns a byte string M.*

Indexing the bits of strings left-to-right starting at 1 and writing $[i]$ for the number i written as an 8-bit byte, the pseudocode is as follows:

Algorithm $\mathsf{Decrypt}_K(C)$
`if` $|C| < 128$ or $|C| \not\equiv 0 \pmod{64}$ `then return` $\perp$
Parse C into $C_0 C_1 \cdots C_m$ with $|C_i| = 64$
`for` $i \leftarrow 1$ `to` m `do` $P_i \leftarrow E_K^{-1}(C_i) \oplus C_{i-1}$
`for` $i \leftarrow 1$ `to` 8 `do`
    `if` $P_m[65-8i..64] = [i]^i$ `then` $P_m \leftarrow P_m[1..65-8i]$, `return` $P_1 \cdots P_m$
`return` $\perp$.

17.3. *Suppose an adversary is given an oracle,* Valid*, that, given a ciphertext C, returns the bit "1" if C is valid, meaning $\mathcal{D}(K, C) \in \{0,1\}^*$, and returns the bit "0" if it is not, meaning $\mathcal{D}(K, C) = \perp$. Show how to use the oracle to decipher a block $Y = E_K(X)$ for an arbitrary eight-byte X. (Hint: all your queries to the* Valid *oracle will be 16 bytes, and I don't mind if you make hundreds or thousands of them.)*

For each string $s \in \{0,1\}^8$, make a query $IV \| Y$ to Valid, with $IV = 0^{56} \| s$. Eventually, the oracle will answer 1, because there exists a string s such that $IV \oplus X$ ends with [1]. However, if the oracle answers 1 then $X \oplus IV$ can end with 01 or 0202 or 030303, and so on. For each $i = 1, \ldots, 7$, let x_i be the 8-byte string obtained by toggling the ith byte of the IV. Make queries

$x_i \| Y$, for $i = 1, \ldots 7$. If the oracle answers 0 to query $x_1 \| Y$ then $IV \oplus X$ ends with $(08)^8$ (Why?). Otherwise, if the oracle answers 1 to query $x_1 \| Y$ but then answers 0 to query $x_2 \| Y$, the string $IV \oplus X$ must end with $(07)^7$, and so on. We can therefore determine the last byte of X. Next, for each $s \in \{0,1\}^8$, make a query $IV \| Y$, with $IV = 0^{48} \| s \| X[57:64] \oplus 02$. The oracle will output 1 if and only if $X \oplus IV$ ends with 0202. We therefore determine the second last byte of X. Repeating this trick to obtain the remaining bytes of X. The total number of queries is $8 \cdot 2^8 + 7 = 2055$.

17.4. *Show how to decrypt any ciphertext* $C = \mathrm{CBC2}(K, M)$ *given a* Valid *oracle.*

Follow algorithm $\mathsf{Decrypt}_K(\cdot)$ to decrypt C. Each time the algorithm requires computing $E_K^{-1}(Y)$, use the algorithm in part (C) to find the answer. If the plaintext has m blocks then the total number of oracle queries is $2055m$.

17.5. *What advice would you give to a security practitioner who was considering the use of* CBC2 *in their networking protocol?*

My advice would be: don't use CBC2. It could be used if one is certain that your application will in no way surface to the adversary which ciphertexts are valid and which are not. But that kind of assurance is a lot to ask for; it would almost certainly be safer to switch to an authenticated-encryption scheme.

Problem 18. *Fix a blockcipher* $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ *and let* $\mathrm{CBCMAC}_K(M)$ *be the CBC MAC, using* E_K*, of a message* M *that is a positive multiple of* n *bits. We have seen that this construction is not secure as a (variable-input-length) MAC.*

18.1. *Consider the construction* $\mathrm{CBCMAC2}_{K\,K'}(M) = \mathrm{CBCMAC}_K(M) \oplus K'$ *where* $K' \in \{0,1\}^n$*. Show that this is a bad MAC—that you can easily forge.*

Here is a simple forging attack. First query $\mathrm{CBCMAC2}_{KK'}(0^n)$ to get $T_1 = E_K(0^n) \oplus K'$. Next query $\mathrm{CBCMAC2}_{KK'}(0^n T_1)$ to get $T_2 = E_K(E_K(0^n) \oplus T_1) \oplus K' = E_K(K') \oplus K'$. Observe that

$$\mathrm{CBCMAC2}(0^n T_1 T_2) = E_K(E_K(K') \oplus T_2) \oplus K' = E_K(K') \oplus K' = T_2 \ .$$

That is, T_2 is the correct tag of $0^n T_1 T_2$, so the adversary that outputs $(0^n T_1 T_2, T_2)$ forges with advantage 1.

18.2. *When strings* x *and* y *are strings with* $|x| > |y|$*, define* $x \oplus y = x \oplus 0^{|x|-|y|} y$*. When* x *is a string and* n *is a fixed value, define* $x10^*$ *as* $x10^i$ *for the smallest* $i \geq 0$ *such that* $|x10^i|$ *is a multiple of* n*. Now consider the construction* $\mathrm{CBCMAC3}_{K\,K'}(M) = \mathrm{CBCMAC}_K(M \oplus K')$ *when* $|M|$ *is a positive multiple of* n*; and* $\mathrm{CBCMAC3}_{K\,K'}(M) = \mathrm{CBCMAC}_K(M10^* \oplus K')$ *otherwise. Here* $|K'| = n$*. Show that* CBCMAC3 *is again a bad MAC—that you can easily forge.*

Here is a simple forging attack. Ask the CBCMAC3 oracle the empty string and learn its MAC T. Now output the pair $(10^{n-1}, T)$, which is an unqueried string, 10^{n-1}, and the correct MAC for it.

Problem 19. *Fix a value* $n \geq 1$ *and the finite field* $\mathbb{F}$ *having* 2^n *points. Represent points in* $\mathbb{F}$ *by* n*-bit strings in the usual way. Now consider the hash function* $H : \mathcal{K} \times (\{0,1\}^n)^+ \to \{0,1\}^n$ *where a string* $M = M_1 \cdots M_m$*, for* $M_i \in \{0,1\}^n$*, hashes to*

$$H_K(M) \quad = \quad M_1 K_1 \ + \ \cdots \ + \ M_m K_m \ + \ K_{m+1} \ .$$

Here $K = (K_1, K_2, \ldots)$ is the key for the hash function, each $K_i \in \mathbb{F}$, and all arithmetic is done in $\mathbb{F}$. A random key from $\mathcal{K}$ is an infinite list of n-bit strings, each uniformly and independently drawn.

19.1. *Prove that H is ε-AU where $\varepsilon = 2^{-n}$.*

Let $M = M_1 \cdots M_m$ and $M' = M'_1 \cdots M'_{m'}$ be distinct messages, each a positive multiple of blocks. We seek to bound $\Pr[H_K(M) = H_K(M')]$. Suppose first that $m \neq m'$, say, without loss of generality, that $m > m'$. In this case,

$$\begin{aligned}\Pr[H_K(M) = H_K(M')] &= \Pr[K_{m+1} = M_1K_1 + \cdots + M_mK_m + M'_1K_1 + \cdots + M'_{m'}K_{m'} + K'_{m'+1}] \\ &= \Pr[K_{m+1} = L]\end{aligned}$$

where $L = L(K_1, \ldots, K_m)$ is a random variable that's independent of K_{m+1}. This value, then, is 2^{-n}. On the other hand, suppose $m = m'$. Then, since $M \neq M'$, there must be some first index i such that $M_i \neq M'_i$. Now

$$\begin{aligned}\Pr[H_K(M) = H_K(M')] &= \Pr[M_1K_1 + \cdots + M_mK_m = M'_1K_1 + \cdots + M'_mK_m] \\ &= \Pr[(M_i + M'_i)K_i = J] \\ &= \Pr[cK_i = J] \\ &= \Pr[K_i = J/c]\end{aligned}$$

where $c \neq 0$ and J is a random variable that is independent of K_i. Thus this value is again 2^{-n}.

19.2. *Show H is not ε-AU, for a small ε, if you omit the last addend in the definition of the hash.*

Without the final addend, we have that $H_K(0^n) = H_K(0^{2n})$ for every K and hence the function is 1-AU (the worst you can get).

19.3. *Name a significant advantage of H and a significant disadvantage of H compared to the polynomial-evaluation hash that I described in class.*

The main advantage of H over the polynomial hash we covered in class is that its ε-value, the probability of collision, doesn't increase with increasing message lengths: it stays constant—and optimal. You might also make the claim that H is more readily *parallelizable* than polynomial-evaluation, which might make it faster to compute in some settings. The main disadvantage of H over the polynomial hash we covered in class is that the key is unboundedly long (or, more precisely, it is n bits longer than the length of the longest message you might have to process).

Note that the conventional computational work for the two hashes is about the same—one multiply and one addition for each block of message, so computational complexity doesn't differentiate them.

Problem 20. *Let E be an n-bit blockcipher. Find a string whose CBC MAC over E you can forge without asking any queries. Explain.*

Let D be the smallest number such that every number less than or equal to 2^n divides it. I claim that we can forge the message $M = 0^{nD}$ using a tag of 0^n. To see that the CBC MAC under E of $(0^n)^D$ is 0^n, notice that whatever the length of the cycle $C = (0^n, E_K(0^n), E_K(E_K(0^n)), \ldots, 0^n)$, it is some number $d \leq 2^n$. When we process the string $(0^n)^D$ using CBC MAC_E, we're going walk this d-cycle an integral number of times because d divides D. That is, we'll go around the cycle D/d times, ending where we started, at 0^n.