CS.202: Intro to Cryptography 11/18/2016

Solution 4

1 CCA insecurity of OFB and CBC modes

Show that OFB and CBC modes of block cipher are not *not* CCA-secure. In each case you can show a CCA attack which sends an encryption challenge pair (m_0, m_1) (note that $|m_0|$ and $|m_1$ must equal, but these messages are not limited to a single block of plaintext!), and then given $c = E(m_b)$ the attack can query the decryption oracle on a *single* ciphertext $c' \neq c$, and decides bit b given the response. In both cases first show the attack, and then argue that your attack is succesful. (If your attack must ask more encryption and decryption queries, that's fine, but you can do it with a single decryption query...)

Solution:

OFB. The CCA attack on OFB is easy because OFB, like CTR, are both of the form $E(k, m) = (IV, G(k, IV) \oplus m)$ where $G(k, \cdot)$ is a variable-range PRF. The two modes construct this PRF $G(k, \cdot)$ differently using the underlying block cipher, but both encryption constructions are actually cases of the basic PRF-based CPA-secure encryption $E(k, m) = (IV, G(k, IV) \oplus m)$ for *some* PRF G. And therefore they are both *xor-malleable* in the same way, and hence not CCA-secure: For any m_0, m_1 s.t. $|m_0| = |m_1| = kn$ for any k, the adversary given $e = E(k, m_b) = (IV, \bar{c})$ can ask the decryption oracle to decrypt $e' = (IV, \bar{c} \oplus \delta)$ for any $\delta \neq 0^{kn}$ (because that makes $e' \neq e$). Note that $m' = D(k, e') = m_b \oplus \delta$, so adversary can compute m_b as $m' \oplus \delta$, and hence decide b.

CBC. CBC encryption is not of the same form as OFB and CTR, but it's still malleable. Note that decryption in CBC is "local" (and hence parallelizable) in the sense that the i-th plaintext block depends only on the i-th and $(i+1)$-st blocks of the ciphertext, because $m_i = c_i \oplus f_k^{-1}(c_{i+1})$. Therefore if you change only i-th block of the target ciphertext, this can affect only the i-th and (i-1)-st block of the plaintext. Therefore given $c = E(k, m^b)$ for m^0 and m^1 s.t. $|m^0| = |m^1| = kn$ for any $k > 1$, the adversary can change e.g. the first block of c (which is the IV), and given $m' = D(k, c')$ for c' resulting from c via such modification of the first block of c, all the blocks of m' will be the same as in m_b except the first. In other words, just from the fact that CBC has parallel decryption procedure it follows that this local change in the ciphertext will *not affect* all the blocks from 2-nd to k-th. Hence the adversary can decide m_b on the basis of these blocks. (Of course m_0 and m_1 must be chosen so that they differ somewhere in these blocks.)

2 Encryption Scheme Attempts

Let $\{F_k\}$ be a PRF family with $K = X = Y = \{0,1\}^n$ for security parameter n. Let G be a PRG s.t. $|G(x)| = 2|x|$ for all x. For each encryption scheme below, state whether it is (a) Indistinguishable, (b) CPA-secure, and (c) CCA-secure, and briefly (but convincingly!) justify your answers. In each case the encryption key space is $\{0,1\}^n$. Assume that a_L, a_R refer to the left half or the right half, respectively, of a bitstring a of even length.

Solution:

A general note on this question is that you should know that CCA security implies CPA security, and CPA security implies IND security, so for each encryption attempt you really have only 4 options:
(1) not IND (and therefore not CPA and not CCA
(2) IND but not CPA (and therefefore also not CCA)
(3) CPA but not CCA (but since it is CPA then it is also IND)
(4) CCA (and therefore also CPA and IND)

(a) Let $M = \{0,1\}^{4(n+1)}$, and let $E(k,m) = (G([k|0]) \oplus m_L, G([k|1]) \oplus m_R)$.

Solution:

This encryption is a case of stream cipher, $E(k,m) = G'(k) \oplus m$ where $G'(k) = (G([k|0])|G([k|1]))$. We know from the previous homework that a stream cipher is IND secure *if and only if* G' is a PRG, therefore any insecure PRG G' makes this E not IND secure. And we also know from the previous homework that this G' is *not* a secure PRG for every PRG G, i.e. that there exist PRG's G which would make G' not a PRG. It follows that the same counterexample PRG G also makes E not IND-secure. And since it's not IND then therefore it's also not CPA and CCA.

Regarding CPA and CCA, you can also notice that E is not CPA (and hence also not CCA) simply because it is deterministic. But this doesn't answer whether E is IND...

(b) Let $M = \{0,1\}^n$, and let $E(k,m) = F(k,k) \oplus m$.

Solution:

Here E is deterministic again, so it's not CPA, and hence also not CCA. But what about IND? Note that $E(k,m) = G(k) \oplus m$ for $G(k)$ defined as $F(k,k)$, so E is IND if and only if $G(k) = F(k,k)$ is a PRG (we knew that from the previous homework...). But is it? $G(k)$ defined as $F(k,c)$ for any *constant* c, would be a PRG, because if A distinguishes $F(k,c)$ for random k from a random value then R can distinguish $F(k,\cdot)$ from a random function: R asks $F(\cdot)$ for its value on c, passes it to A, and returns whatever bit A outputs. R breaks PRF-ness of F if A distinguishes $F(k,c)$ from random, because if $F(\cdot)$ implements $F(k,\cdot)$ A sees $F(k,c)$ and if $F(\cdot)$ is a random function then A sees a random value. But can we use the same logic to show that $F(k,k)$ must be indistinguishable from a random string? Not really: The reduction R breaks because R would have to ask its challenge function $F(\cdot)$ on an argument k, but how can R guess k? The fact that this argument does not work should make you think that maybe it's in fact not necessary that $F(k,k)$ (for a random key k) is pseudorandom for any PRF F. Indeed, how can an efficient adversary query $F(k,\cdot)$ on $x = k$, if k is a random n-bit string, with probability better than negligible in n? If F is a pseudorandom function then it must be hard to predict k after polynomially-many queries to $F(k,\cdot)$, because predicting k implies distinguishing $F(k,\cdot)$ from a random function...

The best way to convince yourself that $G(k) = F(k,k)$ does not have to be a PRG is to construct a counterexample. Consider a PRF $\bar{F}$ whose range are n bit strings and define $F(k,x)$ as 0^n if $x = k$ and as $\bar{F}(k,x)$ for $x \neq k$. If there was an attack A on PRF-ness of F, which outputs 1 more often when it interacts with $F(k,\cdot)$ than when it interacts with a random function, then here's an attack R on PRF-ness of $\bar{F}$:

Whenever A asks for a value on some argument x, R passes x to $\bar{F}$ and passes the value $\bar{F}(x)$ back to A. In addition, R asks $\bar{F}$ for a value v on a random point a, compares it to $v' = \bar{F}(x, a)$ which it computes locally, and stops and outputs 1 if $v = v'$. If that never happens, R stops whenever A stops and outputs whatever A outputs. Let p_0 be the probability A outputs 1 when interacting with a random function while p_1 be the probability A outputs 1 when interacting with $F(k, \cdot)$. Let p'_0 be the probability R outputs 1 when interacting with a random function while p'_1 be the probability R outputs 1 when interacting with $\bar{F}(k, \cdot)$. By assumption, $p_1 - p_0$ is a non-negligible function of n. Note that $|p'_0 - p_0|$ is bounded by the probability that $v = v'$ for any query a when $\bar{F}$ is a random function, but that's bounded by $p(n)/2^n$ where $p(n)$ is the number of PRF queries A makes. Note also that $p'_1 > p_1$, because if $F(\cdot) = F(k, \cdot)$ and A ever queries on $x = k$ then R would output 1, and if A does not make such a query then R's probability of outputing 1 is the same as A's. Therefore $p'_1 - p'_0$ is a non-negligible function of n.

Discussion.

Btw, creating these countexeamples is definitely tricky. For example, how about $F(k, x) = \bar{F}(k \oplus x, x)$? Clearly $F(k, k) = \bar{F}(0^n, k)$, and on any fixed key, e.g. 0^n, a PRF $\bar{F}$ could be a constant function (why? because the probability that a randomly chosen key is equal to 0^n is negligible), e.g. $\bar{F}(0^n, x)$ could be 0^n for all x, in which case $F(k, k) = 0^n$ for all k. Unfortunately, it's not so easy to see if F is a PRF. If you attempted to do a reduction, i.e. given an attacker A against PRF-ness of F you wanted to construct an attacker R against PRF-ness of $\bar{F}$, you'd get into a problem: When A makes a query x, R would have to query $\bar{F}(k \oplus x, \cdot)$, so we'd have to let R interact with $p(n)$ instances of $\bar{F}$, each on a different key k_i but all of these keys would be related i.e. $k_i = k \oplus x_i$ for x_i known to the attacker...

How about $F(k, x) = \bar{F}(k, x) \oplus \bar{F}(k, k)$? Then $F(k, k)$ is also 0^n, but is F a PRF? For any publicly computable pad δ it is true that $F'(k, x) = \bar{F}(k, x) \oplus \delta$ is a PRF. However, the reduction which shows this is (trivially) true, needs to know δ in order to translate attack on F' into an attack on $\bar{F}$, while in the above construction for F the δ pad applied to all values of $\bar{F}$ is not publicly computable, so here again, it's not clear if F is a PRF for every PRF $\bar{F}$...

(c) Let $M = \{0, 1\}^{2n}$, and let $E(k, m) = (r, G(F(k, r)) \oplus m)$ for $r \leftarrow \{0, 1\}^n$.

Solution:

This is not CCA, but it is CPA secure, because it's a particular case of our basic CPA encryption construction (which is not CCA because it's xor-malleable), i.e. $E(k, m) = (r, F'(k, r) \oplus m)$ for some PRF F'. This is a case of this construction for F' constructed as $F'(k, x) = G(F(k, x))$ for a PRG G and PRF F. So we would know that E is CPA *if* this F' is a PRF.

Most students would point that it's "intuitively obvious" that F' is a PRF if G is a PRG and F is a PRF. However, here's how you could really verify this: First, by the PRF-ness property of F, interaction with $F'(k, \cdot)$ is indstinguishable from an interaction with $F''(k, \cdot)$ defined as $F''(x) = G(R(x))$ where R is a random function from n-bit strings to n-bit strings. (The reduction which shows this would interact with attacker A which tries to distinguish between F'' and R, and the reduction would use this A to distinguish between F and R by asking F/R for an output on whatever argument x which A requests, and then running G on this output before handing it back to A.) What this shows is that an interaction with $F'(k, \cdot)$ for random k is indistinguishable from an interaction with algorithm $F''(\cdot)$ where F'' is defined as $F''(x) = G(R(x))$ for R a random function from $\{0, 1\}^n$ to $\{0, 1\}^n$.

But F'' is not yet a random function. It's a random function with G applied to each output. We know that if G is run on random input that its output looks pseudorandom.

An adversary who interacts with $F''(\cdot)$ wil see a polynomial number of instances of G applied to inputs which are randomly chosen each time. So $F''(\cdot)$ *should be* indistinguishable from a random function. And formally you can convince yourself this is true by a series of hybrid arguments: Let H_i be an oracle which responds like a random function to the first i queries of the adversary, and let $p(n)$ be the polynomial bound on the number of queries the adversary makes. Note that H_0 is identical to F'' and that $H_{p(n)}$ is a random function ("RF"). So if A distinguishes F'' from RF with non-negligible advantage ϵ then by the triangle inequality A distinguishes H_i from H_{i-1} for some i iwht non-negligible advantage $\epsilon/p(n)$. Finally, the only difference between H_i and H_{i-1} is that H_i's response to the i-th query is a random $2n$-bit string while H_{i-1}'st response to the i-th query is formed as $G(r)$ for a random n-bit string r. Therefore distinguishing between these two implies an attack on PRG-ness of G.

The reduction which shows this last fact would interact with an A which claims to distinguish between H_i and H_{i-1}. This reduction would get a $2n$-bit value z which is either random or an output of $G(r)$ for random n-bit value r, and it would respond to A's queries as follows: It would answer the first $i-1$ queries of A with random values, it would use z to reply to the i-th query of A, and it would answer each j-th query for $j > i$ by picking random n-bit value r_j and sending back $G(r_j)$ to A. In this way A's view when interacting with this reduction would match A's view of an interaction with H_i if z is random, and it would match A's view of an interaction with H_{i-1} if $z = G(r)$ for a random r. Hence the reduction, by outputting A's decision bit would distinguish between $G(r)$ and a random string, with the same advantage as A distinguishes H_i from H_{i-1}. Therefore, by the PRG-ness of G, $\epsilon/p(n)$ must be negligigble. Therefore ϵ must be neligible. Therefore $F''(\cdot)$ is indistinguishable from a random function. And therefore F is a PRF.

(d) Let $M = \{0,1\}^n$. Let $E(k,m) = (r,v,t)$ where $r \leftarrow \{0,1\}^n$, $v \leftarrow F(k_L,r) \oplus m$, $t \leftarrow F(k_R,v)$, and $(k_L|k_R) \leftarrow G(k)$, and let $D(k,c)$ parse c as a tuple (r,v,t), compute $(k_L|k_R) \leftarrow G(k)$, and output $m = F(k_L,r) \oplus v$ if $t = F(k_R,v)$ and $\perp$ otherwise.

Solution:

This one was tricky. The first thing to notice is that by PRG-ness of G, using (k_L,k_R) computed as $G(k)$ for random n-bit k is no different (up to a negligible factor) to any efficient adversary than using k_L,k_R chosen as two independent random n-bit strings. Therefore the construction would have the same properties (IND, CPA, CCA) if it was modified by taking k_L,k_R as independent random n-bit strings. (Let $k = (k_L,k_R)$.) After this change, the construction is very similar to the construction 4.19 in [KL] of CCA SKE from CPA SKE and unique secure MAC, because $\bar{E}(k_L) = (r, F(k_L,r) \oplus m)$ is a CPA encryption, and we know by construction 4.3 that a PRF is a secure MAC on fixed-length messages. However, there's a difference between this and construction 4.19: It would be the case of construction 4.19 if tag t was computed as $F'(k_R,(r,v))$ for PRF F' which takes $2n$-bit strings. This is because the CPA ciphertext is the *pair* $(r,v) = (r, F(k_L,r) \oplus m)$, and not just the v part of it. In other words, the construction above is a *modification* of the CCA secure construction we know, where we use two keys, k_L for a CPA-secure encryption $\bar{E}$ and k_R for secure MAC M, and the ciphertext is (c,t) where $c = \bar{E}(k_L,m)$ and $t = M(k_R,c)$. The modification is that for $\bar{E}$ defined as above, if $c = (r,v)$ then MAC t is computed only on the v part of the ciphertext...

Some students noticed this difference but argued that this encryption is secure nevertheless, by arguing that because $F(k_R, \cdot)$ is a secure MAC, there's only a negligible probability that a CCA adversary A can ever send a ciphertext (r, v, t) to the decryption oracle $D(k, \cdot)$ s.t. s.t. (1) (r, v, t) is different from every ciphertext (r_i, v_i, t_i) which A gets back as an asnwer to some encryption query m_i to the encryption oracle $E(k, \cdot)$; and (2) the decryption oracle returns $m \neq \perp$ as an answer to this decryption query. If this was true then CCA security of this encryption follows from the CPA security of the "truncated" version of this encryption, i.e. from the CPA security of $E'(k_L, m) = (r, F(k_L, r) \oplus m)$. However, this is *not* true: There is a way to use the decryption oracle and get $m \neq \perp$ for (r, v, t) which does not match any (r_i, v_i, t_i) returned by $E(k, \cdot)$: The decryption oracle will return non-$\perp$ based on just $(v, t) = (v_i, t_i)$ for some i, but r does not have to be equal to r_i.

So the decryption queries cannot be ignored, but how to make them useful? Note that if $t = F(k_R, v)$ then $m = D(k, (r, v, t))$ satisfies $m = F(k_L, r) \oplus v$ (assume that $k = (k_L, k_R)$). We will show how to use this to decrypt the challenge query $(r^*, v^*, t^*) = E(k, m_b^*)$ where (m_0^*, m_1^*) are the two messages chosen for the challenge query. Note that $m_b^* = F(k_L, r^*) \oplus v^*$, so if we managed to learn $F(k_L, r^*)$ then we would compute m_b^* and learn b. Here's an example of how to form a decryption query which would let us learn $F(k_L, r^*)$: Let the adversary first query the encryption oracle $E(k, \cdot)$ on some message m, and let (r, v, t) be the response. Note that $t = F(k_R, v)$. The decryption query which we can then send to $D(k, \cdot)$ to learn $F(k_L, r^*)$ is (r^*, v, t). This is a *valid* ciphertext in this encryption scheme, i.e. the decryption oracle $D(k, \cdot)$ will not reject because $t = F(k_R, v)$ so the MAC in ciphertext (r^*, v, t) is correct![1] Also, this ciphertext query is different from the challenge ciphertext (r^*, v^*, t^*) except if $v^* = v$, which can happen only if $F(k_L, r^*) \oplus F(k_L, r)$ is equal to $m_b^* \oplus m$, but the probability that the latter happens for whatever m_0^*, m_1^*, m we use in the attack is negligible over the choice of k_L because of PRF-ness of F. So except for negligible probability we can send this query to $D(k, \cdot)$, in which case we will receive $m = F(k_L, r^*) \oplus v$ as the decryption. Since we know v, this allows us to compute $F(k_L, r^*)$ as $m \oplus v$, and hence also m_b^* as $m \oplus v \oplus v^*$, and thus learn the bit b.

3 CBC MAC

Consider the fixed-length *"raw CBC"* MAC construction on page 125 of [KL].

Solution:

Let's denote $F(k, \cdot)$ as $R(\cdot)$ throughout (a)-(c) below, i.e. define $R(x)$ as $F(k, x)$ for all x.

(a) Show that this construction is insecure on messages of length ln for any *variable* l.

Solution:

For example you can query the MAC oracle on any n-bit m, get $t = R(m)$, then query the MAC oracle on t, get $t' = R(t)$, and output $m' = (m|t)$ and t' as the MAC forgery.

(b) Show that if in this construction the IV vector t_0 is chosen at random and attached to the tag, instead of being fixed as $t_0 = 0^n$, i.e. if $Tag(k, [m_1|...|m_l]) = (t_0, t_1)$ for

[1] And this is exactly why in the real CCA construction the MAC must be computed on the whole ciphertext, i.e. (r, v) in the case of the above CPA encryption $\bar{E}$, and not just on the v part.

$t_0 \leftarrow \{0,1\}^n$ and $t_i = F(k, m_i \oplus t_{i-1})$ for $i > 0$, then the resulting MAC sheme would be insecure on messages of length ln for any *fixed* l.

Solution:

For example you can query the MAC oracle on any n-bit m, get $(t_0, t_1) = (IV, R(IV \oplus m))$, and output $m' = m \oplus \delta$ and $(t_0 \oplus \delta, t_1)$, which is the MAC forgery for any $\delta \neq m$.

(c) Show that if we include the intermediary block cipher outputs in the tag, i.e. if $Tag(k, [m_1|...|m_l]) = [t_1|...|t_l]$ for $t_0 = 0^n$ and $t_i = F(k, m_i \oplus t_{i-1})$ for $i > 0$, then the resulting MAC scheme would be insecure on messages of length ln for any *fixed* l.

Solution:

For example you can query the MAC oracle on any $2n$-bit (m_1, m_2), get $(t_1, t_2) = (R(m_1), R(t_1 \oplus m_2))$, and output $(t_1 \oplus m_2, t_2 \oplus m_1)$ and (t_2, t_1) as the MAC forgery.

4 HMAC Alternative

Before HMAC was invented, it was common practice to define MAC on variable-sized messages using a Hash Function H as $S(k, m) = H(k|m)$. Show that this is not a secure MAC if H is implemented from a compression function h using a Merkle-Damgard construction (see Figure 5.1, page 158).

Solution:

The attack follows from the observation that if $MD^*([m_1|...|m_B])$ denotes the simplified Merkle-Damgard transform (i.e. the one that omits the length in the last block) applied to message $[m_1|...|m_B]$ then $MD^*([m_1|...|m_B|m_{B+1}]) = h(MD^*([m_1|...|m_B])|m_{B+1})$, i.e. that the MD^* function on message m' which is equal to message $m = [m_1|...|m_B]$ concatenated with an extra block m_{B+1} can be publicly computed from $MD(m)$. Therefore if $S(k, m) = MD^*(k|m)$ then $s' = S(k, m')$ can be publicly computed as $s' = h(s|m_{B+1})$ from $s = S(k, m)$, which shows that $S(k, m) = MD^*(k|m)$ is not a secure MAC.

This observation extends to an attack on MD itself (and not just its simplified version MD^*), because if $s = MD([m_1|...|m_B])$ then $s' = MD([m_1|...|m_B]|\langle nB\rangle|m_{B+1})$ can be computed from s as $s' = h(h(s|m_{B+1})|\langle n(B+1)\rangle)$. Therefore if $S(k, m) = MD(k|m)$ then given tag $s = S(k, m)$ on message $m = [m_1|...|m_B]$, the adversary can create a valid tag $s' = S(k, m')$ on message $m' = [m_1|...|m_B|\langle nB\rangle|m_{B+1}]$ as $s' = h(h(s|m_{B+1})|\langle n(B+1)\rangle)$, which shows that S is not a secure MAC.

5 Merkle-Damgard Transform

Do exercise 5.6 from the textbook, parts (a)-(d). If your answer is yes, *sketch* the proof. If no, demonstrate an attack.

Solution:

All constructions were modifications to the Merkle-Damgard transfom, and the question was whether the modified MD is still collision-resistant.

(a) This modification omitted the input length L encoded in the extra $(B+1)$-st block, i.e., $H'(x) = z_B$ while MD's hash construction is $H(x) = z_{B+1} = h(z_B|L)$.

This is not secure *for every* collision-resistant hash h. To see that trye to apply the argument that H is CRH to argue that H' is CRH: Let x with B blocks and x' with B' blocks be s.t. $x \neq x'$ and $H'(x) = H'(x')$. Let's see if we can get find a collision in h and thus prove that if h is CRH then H' is CRH. Consider $B \neq B'$ and w.l.o.g. $B' = B + t + 1$ for some $t \geq 0$. Denote the blocks of x and x' as $x = (x_1, ..., x_B)$ and $x' = (x'_{-t}, x'_{-t+1}, ..., x'_0, x'_1, ..., x'_B)$. Let $z'_{-t}, z'_{-t+1}, ..., z'_0, z'_1, ..., z'_B$ be the z values on the MD circuit as you evaluate $H(x')$, and let $z_1, ..., z_B$ be the corresponding values on the MD circuit as you evaluate $H(x)$. By the construction we have that if $H(x) = H(x')$ then either $(z_{B-1}|x_B) = (z'_{B-1}, x'_B)$ or $(z_B|x_B), (z'_B, x'_B)$ form a collision in h. In the latter case we have a collision in h so consider the former case, but this case implies that $z_{B-1} = z'_{B-1}$. So this way we can go back the change exactly as in the original security argument for Merkle-Damgard, concluding that $(x_1, ..., x_B) = (x'_1, ..., x'_B)$, until we hit the step $IV = z'_0$. Here we have that $h(z'_{-1}, x'_0) = z'_0 = IV$, but this is not a *collision* in h: We have one preimage of IV under h, namely $z_{-1}|x'_0$, but a single pre-image of some fixed value (here the Merkle-Damgard constant IV) is not a collision. So it's not clear how to argue that this variant of MD is secure. Indeed, any collision-resistant h has to have *some* value on input $z_{-1}|x'_0$: If IV happens to be that value, i.e. we just define IV as $h(z_{-1}|x'_0)$, then this will lead to an attack on the modified MD.

(b) This modification defined $H'(x) = (z_B|L)$ instead of $H(x) = h(z_B|L)$.

This is as secure as the original Merkle-Damgard. It's simple to argue: Suppose that you found a collision in H', i.e. x, x' s.t. $x{\neq}x'$ and $H'(x) = H'(x)$. It follows by H' construction that $z'_B|L' = z_B|L$. But since the real Merkle-Damgard H for these inputs would output $H(x) = h(z_B|L)$ and $H(x') = h(z'_B|L')$ it follows that $H(x) = H(x')$, and therefore we would have a collision in H. This contradicts the collision-resistance property of Merkle-Damgard hash H, and hence it follows that H' must also be collision-resistant.

(c) This modification changed the contruction by starting from $z_0 = x_1$ instead of $z_0 = IV$ where IV is the MD's constant, i.e. $z_1 = x_1$, and then $z_i = h(z_{i-1}|x_i)$ for $i = 2, ..., B+1$, with $x_{B+1} = L$, and $H'(x) = z_{B+1}$.

This is secure: The same security argument as in the original Merkle-Damgard shows that if $H(x) = H(x')$ then $|x| = |x'|$, and that all the blocks are the same too, all the way down to $h(z_2|x_2) = h(z'_2|x'_2)$ which implies that $x_2 = x'_2$ and $z_2 = z'_2$, and the last fact implies $h(x_1|x_2) = h(x'_1|x'_2)$ so either $x_1|x_2 = x'_1|x'_2$ or there is a collision in h.

(d) This modification used L instead of IV, i.e. $z_0 = L$, and then $z_i = h(z_{i-1}|x_i)$ for $i = 1, ..., B$, and $H'(x)$ returns z_B.

This is insecure in general. Consider the same counterexample as in part (a). If $H(x_1|...|x_B) = H(x'_{-t}|...|x'_0|x'_1|...|x'_B)$ then we get that $x'_1|...|x'_B = x_1|...|x_B$ (or we have a collision in h at some point), and that $z'_0 = z_0 = L$, i.e. that $h(z'_{-1}, x'_0) = z'_0 = L$, but that again just means that we have found a pre-image of function h on some particular value, namely L, but that's not a collision in h. If we want x' to have just one more block than x we need that $h(L'|x'_0) = L$ for $L' = L + 1$. This is a strange behavior for h to have, and if h was a random function the probability of this behavior would be negligible, but all we know about h is that it is collision-resistant, and that is not contradicted if $H(L+1|x'_0) = L$ for some x'_0, L.