

Additional hints for Assignment 1

February 26, 2022

1a. Write down explicitly what $\mathbf{Adv}_{\text{MAC}}^{\text{unf}}$ is, and use a triangle inequality to replace F_K by something convenient (by what exactly, should be interpretable from the right hand side of the equation).

1b. Write down explicitly the distinguishing model, i.e., to what functions a distinguisher \mathcal{D} has access to when considering $\mathbf{Adv}_{\text{MAC}}^{\text{prf}}(\mathcal{D})$, and investigate what is a difference between these functions that \mathcal{D} could exploit.

2b. Try to find 4 MAC evaluations of the function that “bitwise sum to 0”. Note that, once you know this, you only need to make 3 of those MAC queries and the 4th one would be the forgery.

3. A clarification on the notation: in this authenticated encryption scheme, the input to the block cipher is $(K, N \| M)$ and the output $C \| T$, where “ $\|$ ” stands for concatenation.

3b. The assignment basically asks you to fill the dots. Hint: write down explicitly what the $\Delta_{\mathcal{D}}(\dots; \dots)$ ’s are in terms of probabilities: $\Delta_{\mathcal{D}}(\dots; \dots) = |\mathbf{Pr}(\dots) - \mathbf{Pr}(\dots)|$.

3c. At this point, all that \mathcal{D} is doing, is comparing the outputs of two different types of random functions (try see this!). The bound for this has been seen before, already at Introduction to Cryptography.

4. The attacker model is as follows.

- It can make online queries to \tilde{E}_K . Here, it can make a forward query (T, M) to obtain C , or it can make an inverse query (T, C) to obtain M .
- It can make offline evaluations of the primitive, in this case E . Here, it can make a forward evaluation (L, X) for key L and data input X to receive data output Y , or it can make an inverse evaluation (L, Y) to obtain X .

The attack of this exercise actually works by only making forward queries to \tilde{E}_K and forward evaluations of E .

If you don’t see the attack at first sight, first consider the case of 1 construction query (e.g., for tweak $T = 0^k$) and *explicitly* write down how you would do a key recovery attack. Then, expand the case to 2 construction queries, etc.

(See other side for question 5.)

5. In this exercise, we work with the finite field $\text{GF}(2^{128})$, and we interpret strings $a = a_{127}a_{126} \dots a_2a_1a_0 \in \{0,1\}^{128}$ as polynomials

$$a(X) = a_{127}X^{127} + a_{126}X^{126} + \dots + a_1X + a_0,$$

which we add and multiply (as you would do with numbers). We also reduce modulo the polynomial $q(X) = X^{128} + X^7 + X^2 + X + 1$. If you have not yet worked with finite fields in much detail before, this may require a bit of extra explanation. A detailed explanation can be found in Chapter 21 of Intro2Crypto lecture notes.

In general, it is easy to work in a finite field of prime size p , such as 2 or 13. Taking $p = 13$ as an example, it works to just take \mathbb{Z} modulo 13 to get a finite field: you can add and multiply numbers and reduce modulo 13, and you never get that $a \cdot b = 0$ if neither a nor b is zero. Even better, you get an inverse a^{-1} for every non-zero a , e.g., 6 has the inverse 11 because $6 \cdot 11 = 66$ and $66 \bmod 13$ is 1. If we would have done computations modulo a number such as 15, we immediately get problems with $3 \cdot 5 = 15 = 0 \bmod 15$... it doesn't work so nicely!

For numbers such as $8 = 2^3$ or $81 = 3^4$ however, finite fields do exist, that is, for *prime powers* p^k . But they're not "just" working modulo p^k . (Do you see why we get the same issues as we got for 15?). To build a field with p^k elements (such as our case of 2^{128}) we have to switch to the polynomials we described before: basically, we have the 2^{128} polynomials (of degree 127 and less) as the elements of our fields, and you can think of them as numbers: you can multiply and add them, keeping the variable X as an unknown. And as we noted above, we can switch from string-representation to polynomial-representation and back.

So we interpret the string as a polynomial, perform the computations there, and switch back to the string when we computed what we want. This leaves us with the weird polynomial $q(X)$ with which we reduce, which looks a bit arbitrary. Why specifically use this polynomial to reduce? Let's take a look at a smaller example: $\text{GF}(2^4)$ where we work with polynomials of degree 3 and below, and reduce modulo $q(X) = X^4 + X + 1$.

We would get for example $X^3 \cdot X^3 = X^6$ and reducing modulo $q(X)$ just means interpreting $X^4 + X + 1$ as 0, or equivalently X^4 as $-X - 1$ (do you see why?). So $X^6 = X^2 \cdot X^4 = X^2 \cdot (-X - 1) = -X^3 - X^2$ modulo $q(X)$. And just as we had with \mathbb{Z} modulo a number p , we need the polynomial $q(X)$ to have some good property for this to work, namely, it shouldn't factor into smaller polynomials! If it did, like $X^4 - 1 = (X^2 + 1)(X^2 - 1)$, those two smaller polynomials would multiply and become 0, just like $3 \cdot 5 = 0$ modulo 15 before. So, we can take a specific polynomial q of degree 4 that has this property that it cannot be factored, and work with polynomials of degree below 4 and reduce modulo q . You may wonder now, don't we get different results if we take different polynomials q to reduce with? We do, so we get different fields of order 2^4 in this case, one per each q . But we talk about **the** field $\text{GF}(2^4)$? Interestingly, (see Theorem 21.3.19) all of the possible fields we get for such polynomials behave in the same way: they are isomorphic. So when we talk about **the** field $\text{GF}(2^4)$, we just mean any such field, and it doesn't really matter which one because they are so similar!

In the case of exercise 5, you can check (theoretically, but not easy in practice) that $q(X)$ has the nice property that it cannot be factored into smaller polynomials, hence this choice works: we get a field of 2^{128} elements, which we can interpret as polynomials to make calculations easier to understand.