2 Mathematics for Cryptographers II

2.1 Basics of Security Analysis

In many cases, technical papers include the analysis on the key question: "How secure is this cryptographic algorithm?" or rather "Why this cryptographic algorithm is secure?". In this section, we will shortly describe the notation and typical construction for justifying the security of cryptographic algorithms.

Typically, the cryptographic security is defined in a form of a game between the adversary (who we call $\mathcal{A}$) and the challenger (who we call $\mathcal{C}h$). The adversary is trying to break the security of the cryptographic algorithm using arbitrary (but still efficient) protocol, while the challenger is following a simple, fixed protocol. The game is played in a form of a challenge, where the adversary is given some information and is asked to perform some task. The security of the cryptographic algorithm is defined based on the probability of the adversary to win the game.

2.1.1 Cipher Semantic Security

Let us get into specifics. Suppose that we want to specify that the encryption scheme is secure. Recall that cipher $\mathcal{E} = (E, D)$ over the space $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ (here, $\mathcal{K}$ is the space containing all possible keys, $\mathcal{M}$ − all possible messages and $\mathcal{C}$ − all possible ciphers) consists of two efficiently computable methods:

- $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ − encryption method, that based on the provided message $m \in \mathcal{M}$ and key $k \in \mathcal{K}$ outputs the cipher $c = E(k, m) \in \mathcal{C}$.
- $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ − decryption method, that based on the provided cipher $c \in \mathcal{C}$ and key $k \in \mathcal{K}$ outputs the message $m = D(k, c) \in \mathcal{M}$.

Of course, we require the **correctness**:

$$(\forall k \in \mathcal{K})\,(\forall m \in \mathcal{M}) : \{D(k, E(k, m)) = m\} \tag{24}$$

Now let us play the following game between adversary $\mathcal{A}$ and challenger $\mathcal{C}h$:

1. $\mathcal{A}$ picks any two messages $m_0, m_1 \in \mathcal{M}$ on his choice.
2. $\mathcal{C}h$ picks a random key $k \xleftarrow{R} \mathcal{K}$ and random bit $b \xleftarrow{R} \{0, 1\}$ and sends the cipher $c = E(k, m_b)$ to $\mathcal{A}$.
3. $\mathcal{A}$ is trying to guess the bit b by using the cipher c.
4. $\mathcal{A}$ outputs the guess $\hat{b}$.

Now, what should happen if our encryption scheme is secure? The adversary should not be able to guess the bit b with a probability significantly higher than $1/2$ (a random guess). Formally, define the **advantage** of the adversary $\mathcal{A}$ as:

$$\mathsf{SSAdv}[\mathcal{E}, \mathcal{A}] := \left| \Pr\left[\hat{b} = b\right] - \frac{1}{2} \right| \tag{25}$$

We say that the encryption scheme is **semantically secure**[4] if for any efficient adversary $\mathcal{A}$ the advantage $\mathsf{SSAdv}[\mathcal{A}]$ is negligible. In other words, the adversary cannot guess the bit b with a probability significantly higher than $1/2$.

---

[4]This version of definition is called a **bit-guessing** version.
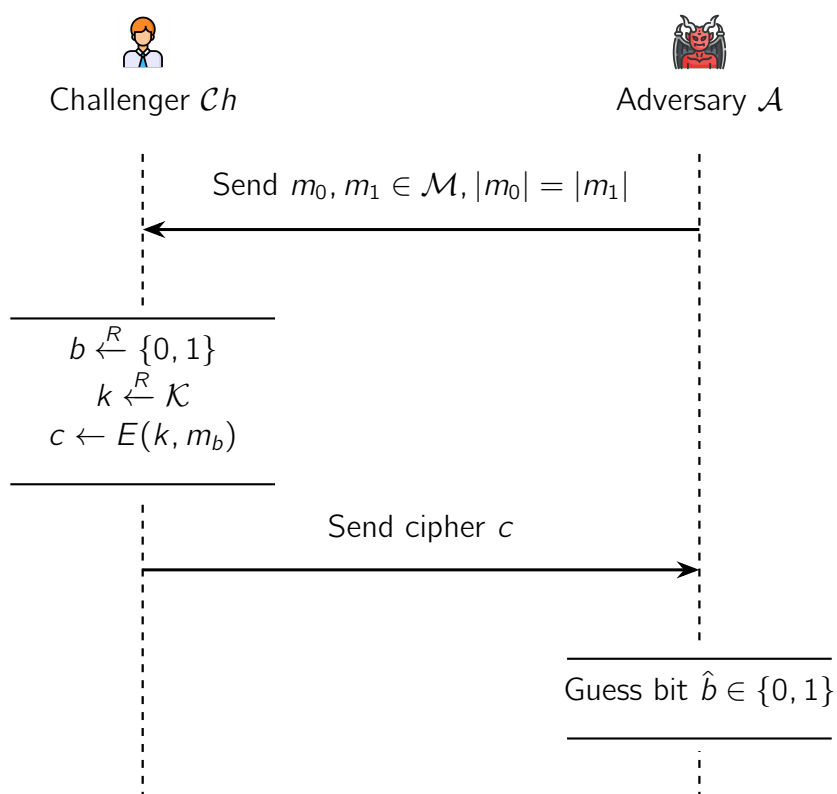
Figure 5: The game between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}h$ for defining the semantic security.

Now, what negligible means? Let us give the formal definition!

Definition 2.1. A function $f : \mathbb{N} \to \mathbb{R}$ is called **negligible** if for all $c \in \mathbb{R}_{>0}$ there exists $n_c \in \mathbb{N}$ such that for any $n \geq n_c$ we have $|f(n)| < 1/n^c$.

The alternative definition, which is problably easier to interpret, is the following.

Theorem 2.2. A function $f : \mathbb{N} \to \mathbb{R}$ is **negligible** if and only if for any $c \in \mathbb{R}_{>0}$, we have

$$\lim_{n \to \infty} f(n)n^c = 0 \tag{26}$$

Example. The function $f(n) = 2^{-n}$ is negligible since for any $c \in \mathbb{R}_{>0}$ we have

$$\lim_{n \to \infty} 2^{-n}n^c = 0 \tag{27}$$

The function $g(n) = \frac{1}{n!}$ is also negligible for similar reasons.

Example. The function $h(n) = \frac{1}{n}$ is not negligible since for $c = 1$ we have

$$\lim_{n \to \infty} \frac{1}{n} \times n = 1 \neq 0 \tag{28}$$

Well, that is weird. For some reason we are considering a function the depends on some natural number n, but what is this number?

Typically, when defining the security of the cryptographic algorithm, we are considering the security parameter λ (e.g., the length of the key). The function is negligible if the probability of the adversary to break the security of the cryptographic algorithm is decreasing with the increasing of the security parameter λ. Moreover, we require that the probability of the adversary to break the security of the cryptographic algorithm is decreasing faster than any polynomial function of the security parameter λ.

So all in all, we can define the semantic security as follows.

Definition 2.3. The encryption scheme $\mathcal{E}$ with a security paramter $\lambda \in \mathbb{N}$ is **semantically secure** if for any efficient adversary $\mathcal{A}$ we have:

$$\left| \Pr\left[b = \hat{b} \,\middle|\, \begin{array}{c} m_0, m_1 \leftarrow \mathcal{A}, \; k \xleftarrow{R} \mathcal{K}, \; b \xleftarrow{R} \{0,1\} \\ c \leftarrow E(k, m_b) \\ \hat{b} \leftarrow \mathcal{A}(c) \end{array} \right] - \frac{1}{2} \right| < \mathsf{negl}(\lambda) \tag{29}$$

Do not be afraid of such complex notation, it is quite simple. Notation $\Pr[A \mid B]$ means "the probability of A, given that B occurred". So our inner probability is read as "the probability that the guessed bit $\hat{b}$ equals b given the setup on the right". Then, on the right we define the setup: first we generate two messages $m_0, m_1 \in \mathcal{M}$, then we choose a random bit b and a key k, cipher the message m_b, send it to the adversary and the adversary, based on provided cipher, gives $\hat{b}$ as an output. We then claim that the probability of the adversary to guess the bit b is

close to 1/2.

Let us see some more examples of how to define the security of certain crypographic objects.

2.1.2 Discrete Logarithm Assumption (DL)

Now, let us define the fundamental assumption used in cryptography formally: the **Discrete Logarithm Assumption** (DL).

Definition 2.4. Assume that $\mathbb{G}$ is a cyclic group of prime order r generated by $g \in \mathbb{G}$. Define the following game:

1. Both challenger $\mathcal{Ch}$ and adversary $\mathcal{A}$ take a description $\mathbb{G}$ as an input: order r and generator $g \in \mathbb{G}$.
2. $\mathcal{Ch}$ computes $\alpha \xleftarrow{R} \mathbb{Z}_r$, $u \leftarrow g^\alpha$ and sends $u \in \mathbb{G}$ to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs $\hat{\alpha} \in \mathbb{Z}_r$.

We define $\mathcal{A}$'s **advantage in solving the discrete logarithm problem in** $\mathbb{G}$, denoted as $\mathsf{DLadv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{\alpha} = \alpha$.

Definition 2.5. The **Discrete Logarithm Assumption** holds in the group $\mathbb{G}$ if for any efficient adversary $\mathcal{A}$ the advantage $\mathsf{DLadv}[\mathcal{A}, \mathbb{G}]$ is negligible.

Informally, this assumption means that given u, it is very hard to find α such that $u = g^\alpha$. But now we can write down this formally!

2.1.3 Computational Diffie-Hellman (CDH)

Another fundamental problem in cryptography is the **Computational Diffie-Hellman** (CDH) problem. It states that given g^α, g^β it is hard to find $g^{\alpha\beta}$. This property is frequently used in the construction of cryptographic protocols such as the Diffie-Hellman key exchange.

Let us define this problem formally.

Definition 2.6. Let $\mathbb{G}$ be a cyclic group of prime order r generated by $g \in \mathbb{G}$. Define the following game:

1. Both challenger $\mathcal{Ch}$ and adversary $\mathcal{A}$ take a description $\mathbb{G}$ as an input: order r and generator $g \in \mathbb{G}$.
2. $\mathcal{Ch}$ computes $\alpha, \beta \xleftarrow{R} \mathbb{Z}_r$, $u \leftarrow g^\alpha$, $v \leftarrow g^\beta$, $w \leftarrow g^{\alpha\beta}$ and sends $u, v \in \mathbb{G}$ to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs $\hat{w} \in \mathbb{G}$.

We define $\mathcal{A}$'s **advantage in solving the computational Diffie-Hellman problem in** $\mathbb{G}$, denoted as $\mathsf{CDHadv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{w} = w$.

Definition 2.7. The **Computational Diffie-Hellman Assumption** holds in the group $\mathbb{G}$ if for any efficient adversary $\mathcal{A}$ the advantage $\mathsf{CDHadv}[\mathcal{A}, \mathbb{G}]$ is negligible.

2.2 Decisional Diffie-Hellman (DDH)

Now, we loosen the requirements a bit. The **Decisional Diffie-Hellman** (DDH) problem states that given $g^\alpha, g^\beta, g^{\alpha\beta}$ it is "hard" to distinguish $g^{\alpha\beta}$ from a random element in $\mathbb{G}$. For-

mally, we define this problem as follows.

> **Definition 2.8.** Let $\mathbb{G}$ be a cyclic group of prime order r generated by $g \in \mathbb{G}$. Define the following game:
>
> 1. Both challenger $\mathcal{Ch}$ and adversary $\mathcal{A}$ take a description $\mathbb{G}$ as an input: order r and generator $g \in \mathbb{G}$.
> 2. $\mathcal{Ch}$ computes $\alpha, \beta, \gamma \xleftarrow{R} \mathbb{Z}_r, u \leftarrow g^{\alpha}, v \leftarrow g^{\beta}, w_0 \leftarrow g^{\alpha\beta}, w_1 \leftarrow g^{\gamma}$. Then, $\mathcal{Ch}$ flips a coin $b \xleftarrow{R} \{0,1\}$ and sends u, v, w_b to $\mathcal{A}$.
> 3. The adversary $\mathcal{A}$ outputs the predicted bit $\hat{b} \in \{0,1\}$.
>
> We define $\mathcal{A}$'s **advantage in solving the Decisional Diffie-Hellman problem in** $\mathbb{G}$, denoted as DDHadv$[\mathcal{A}, \mathbb{G}]$, as
>
> $$\text{DDHadv}[\mathcal{A}, \mathbb{G}] := \left| \Pr\left[b = \hat{b} \right] - \frac{1}{2} \right| \tag{30}$$

Now, let us break this assumption for some quite generic group! Consider the following example.

> **Theorem 2.9.** Suppose that $\mathbb{G}$ is a cyclic group of an even order. Then, the Decision Diffie-Hellman Assumption does not hold in $\mathbb{G}$. In fact, there is an efficient adversary $\mathcal{A}$ that can distinguish $g^{\alpha\beta}$ from a random element in $\mathbb{G}$ with an advantage $1/4$.

Proof. If $|\mathbb{G}| = 2n$ for $n \in \mathbb{N}$, it means that we can split the group into two subgroups of order n, say, $\mathbb{G}_1$ and $\mathbb{G}_2$. The first subgroup consists of elements in a form g^{2k}, while the second subgroup consists of elements in a form g^{2k+1}.

Now, if we could efficiently determine, based on group element $g \in \mathbb{G}$, whether $g \in \mathbb{G}_1$ or $g \in \mathbb{G}_2$, we essentially could solve the problem. Fortunately, there is such a method! Consider the following lemma.

> **Lemma 2.10.** Suppose $u = g^{\alpha}$. Then, α is even if and only if $u^n = 1$.

Proof. If α is even, then $\alpha = 2\alpha'$ and thus

$$u^n = (g^{2\alpha'})^n = g^{2n\alpha'} = (g^{2n})^{\alpha'} = 1^{\alpha'} = 1 \tag{31}$$

Conversely, if $u^n = 1$ then $u^{\alpha n} = 1$, meaning that $2n \mid \alpha n$, implying that α is even. Lemma is proven.

Now, we can construct our adversary $\mathcal{A}$ as follows. Suppose $\mathcal{A}$ is given (u, v, w). Then,

1. Based on u, get the parity of α, say $p_\alpha \in \{\text{even}, \text{odd}\}$.
2. Based on v, get the parity of β, say $p_\beta \in \{\text{even}, \text{odd}\}$.
3. Based on w, get the parity of γ, say $p_\gamma \in \{\text{even}, \text{odd}\}$.
4. Calculate $p'_\gamma \in \{\text{even}, \text{odd}\}$ — parity of $\alpha\beta$.
5. Return $\hat{b} = 0$ if $p'_\gamma = p_\gamma$, and $\hat{b} = 1$, otherwise.

Suppose γ is indeed $\alpha \times \beta$. Then, condition $p'_\gamma = p_\gamma$ will always hold. If γ is a random element, then the probability that $p'_\gamma = p_\gamma$ is $1/2$. Therefore, the probability that $\mathcal{A}$ will guess the bit b correctly is $3/4$, and the advantage is $1/4$ therefore. ∎

2.2.1 Why this is needed?

Typically, it is impossible to prove the predicate "for every efficient adversary $\mathcal{A}$ this probability is negligible" and therefore we need to make assumptions, such as the Discrete Logarithm Assumption or the Computational Diffie-Hellman Assumption. In turn, proving the statement "if X is secure then Y is also secure" is manageable and does not require solving any fundamental problems. So, for example, knowing that the probability of the adversary to break the Diffie-Hellman assumption is negligible, we can prove that the Diffie-Hellman key exchange is secure.

2.3 Basic Number Theory

2.3.1 Primes

Primes are often used when doing almost any cryptographic computation. A prime number is a natural number ($\mathbb{N}$) that is not a product of two smaller natural number. In other words, the prime number is divisible only by itself and 1. The first primes are: 2, 3, 5, 7, 11...

2.3.2 Deterministic prime tests

A primality test is deterministic if it outputs `True` when the number is a prime and `False` when the input is composite with probability 1. An example of a deterministic prime test is `Trial_Division_Test`. Here is an example implementation in Rust:

```rust
fn is_prime(n: u32) -> bool {
    let square_root = (n as f64).sqrt() as u32;

    for i in 2.. = square_root {
        if n % i == 0 {
            return false;
        }
    }

    true
}
```

Deterministic tests often lack efficiency. For instance, even with square root optimization, the asymptotic complexity is $O(\sqrt{N})$. While further optimizations are possible, they do not change the overall asymptotic complexity.

In cryptography, N can be extremely large — 256 bits, 512 bits, or even 6144 bits. An algorithm is impractical when dealing with such large numbers.

2.3.3 Probabilistic prime tests

A primality test is probabilistic if it outputs `True` when the number is a prime and `False` when the input is composite with probability less than 1. Such test is often called a pseudoprimality test. Fermat Primality and Miller-Rabin Primality Tests are examples of probabilistic primality test. Both of them use the idea of **Fermat's Little Theorem**:

Theorem 2.11. Let p be a prime number and a be an integer not divisible by p. Then $a^{p-1} - 1$ is always divisible by p: $a^{p-1} \equiv 1 \pmod{p}$

The key idea behind the Fermat Primality Test is that if for some a not divisible by n we have $a^{n-1} \not\equiv 1 \pmod{n}$ then n is definitely NOT prime. Athough, with such an approach, we might get a false positive, as you cannot state for sure that n is prime. For example, consider $n = 15$ and $a = 4$. $4^{15-1} \equiv 1 \pmod{15}$, but $n = 15 = 3 \cdot 5$ is composite. To solve this issue, a is picked many times, decreasing the chances of a false positive. The probability that a composite number is mistakenly called prime for k iterations is $2^{-k} = \dfrac{1}{2^k}$.

There exists a problem with such an algorithm in the form of **Carmichael numbers**, which are numbers that are Fermat pseudoprime to all bases. To put it simply, no matter how many times you check whether the number is prime using this type of primality test, it will always stay positive, even though the number is composite. The good thing is that Carmichael numbers are pretty rare. The bad thing is that there are infinitely many of them.

Even though this algorithm is probabilistic (which does not guarantee the correctness of the output) and has a vulnerability in the form of *Carmichael numbers*, it runs with an asymptotic complexity $O(\log^3 n)$. This is much better for large numbers and is often used in cryptography. Here is a pseudocode implementation of this algorithm:

```
1  # n = number to be tested for primality
2  # k = number of times the test will be repeated
3  def is_prime(n, k):
4      i = 1
5      while i <= k:
6          a = rand(2, n - 1)
7
8          if a^(n - 1) != 1 (mod n):
9              return False
10
11         i++
12
13     return True
```

Miller-Rabin primality test, is a more advanced form of Fermat primality test. The main difference is it is not vulnerable to *Carmichael numbers*, which makes it much better to use in practice.

2.3.4 Greatest Common Divisor

Greatest common divisor (GCD) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.

> **Example.** $gcd(8, 12) = 4$, $gcd(3, 15) = 3$, $gcd(15, 10) = 5$.

Computing GCD using Euclid's algorithm. The is based on the fact that, given two positive integers a and b such that $a > b$, the common divisors of a and b are the same as the common divisors of $a - b$ and b. It can be observed, that it can be further optimized, by using $a \bmod b$, instead of $a - b$. For example, $gcd(26, 8) = gcd(18, 8) = gcd(10, 8) = gcd(2, 8)$ can be optimized to $gcd(26, 8) = gcd(26 \pmod 8, 8) => gcd(2, 8)$ Algorithm can be implemented using recursion. Base of the recursion is $gcd(a, 0) = a$.

```
1       int gcd(a, b):
2        if (b == 0):
3            return a
4        return gcd(b, a % b)
```

Provided algorithm work with $O(log(N))$ asymptotic complexity.

2.3.5 Least common multiple

Least common multiple (LCM) of two integers a and b, is the smallest positive integer that is divisible by both a and b.

The least common multiple can be computed from the greatest common divisor with the formula: $lcm(a, b) = \frac{|ab|}{gcd(a,b)}$

```
1       int lcm(a, b):
2        return a * (b / gcd(a, b))
```

2.3.6 Modular inverse

Modular multiplicative inverse of an integer a is an integer b such that $a \cdot b \equiv 1 \pmod{m}$. In prime fields it is commonly used as a division operation.

One of the ways to compute the modular inverse is by using Euler's theorem:
$a^{\phi(m)} \equiv 1 \pmod{m}$, where ϕ is Euler's totient function.
For prime numbers, where $\phi(m) = m - 1$:
$a^{m-2} \equiv a^{-1} \pmod{m}$.

```
1       a_inverse = powmod(a, m-2, m) # where powmod(base, power,
          ↪ modulus)
```

2.3.7 Reed-Solomon codes

Reed-Solomon codes allows to restore lost or corrupted data, implement threshold secret sharing and is used in some ZK protocols. Given a vector of data V a polynomial P is constructed using Lagrange interpolation. Polynomial with degree n can be uniquely defined using $(n + 1)$ unique points. Defining more points on the same polynomial add a redundancy, which can be used to restore the polynomial even if some points are missing. Common choices for a set of evaluation points include $0, 1, 2, ..., n - 1$.

The error-correcting ability of a Reed-Solomon code is $n - k$, the measure of redundancy in the block. If the locations of the error symbols are not known in advance, then a Reed-Solomon code can correct up to $n - k/2$ erroneous symbols, i.e., it can correct half as many errors as there are redundant symbols added to the block.

2.3.8 Schwartz-Zippel Lemma

Lemma 2.12. Let $\mathbb{F}$ be a field. Let $f(x_1, x_2, ..., x_n)$ be a polynomial of total degree d. Suppose that f is not the zero polynomial. Let S be a finite subset of $\mathbb{F}$. Let $r_1, r_2, ...r_n$ be chosen at random uniformly and independently from S. Then the probability that $f(r_1, r_2, ..., r_n) = 0$

is $\leq \frac{d}{|S|}$.

Example. Let $F = \mathbb{F}_3$, $f(x) = x^2 - 5x + 6$, $S = F$, $r \xleftarrow{R} \mathbb{F}_3$.
Schwartz-Zippel lemma says that the probability that $f(r) = 0$ is $\leq \frac{2}{3}$.

Given two polynomials P, Q with degree d in a field $\mathbb{F}_p$, for $r \xleftarrow{R} \mathbb{F}_3$: $\Pr[P(r) == Q(r)] \leq \frac{d}{p}$. For large fields, where $\frac{d}{p}$ is negligible, this property allows to succinctly check the equality of polynomials. Let $H(x) := P(x) - Q(x)$. Than for each $P(x) = Q(x) \to H(x) = 0$. Applying Schwartz-Zippel lemma, the probability of $H(x) = 0$ for $x \xleftarrow{R} \mathbb{F}$ is $\leq \frac{d}{|S|}$.

2.4 Exercises

Exercise 1. Suppose that for the given cipher with a security parameter λ, the adversary $\mathcal{A}$ can deduce the least significant bit of the plaintext from the ciphertext. Recall that the advantage of a bit-guessing game is defined as $\mathsf{SSAdv}[\mathcal{A}] = \left| \Pr[b = \hat{b}] - \frac{1}{2} \right|$, where b is the randomly chosen bit of a challenger, while $\hat{b}$ is the adversary's guess. What is the maximal advantage of $\mathcal{A}$ in this case?

Hint: The adversary can choose which messages to send to challenger to further distinguish the plaintexts.

a) 1

b) $\frac{1}{2}$

c) $\frac{1}{4}$

d) 0

e) Negligible value ($\mathsf{negl}(\lambda)$).

Exercise 2. Consider the cipher $\mathcal{E} = (E, D)$ with encryption function $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ over the message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and key space $\mathcal{K}$. We want to define the security that, based on the cipher, the adversary $\mathcal{A}$ cannot restore the message (*security against message recovery*). For that reason, we define the following game:

1. Challenger chooses random $m \xleftarrow{R} \mathcal{M}$, $k \xleftarrow{R} \mathcal{K}$.

2. Challenger computes the ciphertext $c \leftarrow E(k, m)$ and sends to $\mathcal{A}$.

3. Adversary outputs $\hat{m}$, and wins if $\hat{m} = m$.

We say that the cipher $\mathcal{E}$ is secure against message recovery if the **message recovery advantage**, denoted as $\mathsf{MRadv}[\mathcal{A}, \mathcal{E}]$ is negligible. Which of the following statements is a valid interpretation of the message recovery advantage?

a) $\mathsf{MRadv}[\mathcal{A}, \mathcal{E}] := \left| \Pr[m = \hat{m}] - \frac{1}{2} \right|$

b) $\mathsf{MRadv}[\mathcal{A}, \mathcal{E}] := |\Pr[m = \hat{m}] - 1|$.

c) $\mathsf{MRadv}[\mathcal{A}, \mathcal{E}] := \Pr[m = \hat{m}]$

d) $\mathsf{MRadv}[\mathcal{A}, \mathcal{E}] := \left| \Pr[m = \hat{m}] - \frac{1}{|\mathcal{M}|} \right|$

Exercise 3. Suppose that f and g are negligible functions. Which of the following functions is not neccessarily negligible?

a) $f + g$

b) $f \times g$

c) $f - g$

d) f/g

e) $h(\lambda) := \begin{cases} 1/f(\lambda) & \text{if } 0 < \lambda < 100000 \\ g(\lambda) & \text{if } \lambda \geq 100000 \end{cases}$

Exercise 4. Suppose that $f \in \mathbb{F}_p[x]$ is a d-degree polynomial with d **distinct** roots in $\mathbb{F}_p$. What is the probability that, when evaluating f at n random points, the polynomial will be zero at all of them?

a) Exactly $(d/p)^n$.

b) Strictly less that $(d/p)^n$.

c) Exactly nd/p.

d) Exactly d/np.