Problem 1: Pseudorandom Fun(ctions)

We know that we want the output of $F'_s(x)$ to be of the form $y_0 y_1 \ldots y_{k-1}$, where $y_{i-1}$ represents the $i$th bit of the output $y$. To construct $F'_s(x)$ from $F_s(z)$, we can set $y_i = F_s(x \circ i)$ for all $0 \leq i < k$. Because there are $k$ bits in the output, it will take $\lceil \log k \rceil$ bits to represent $i$ and so the input $x \circ i$ will have $k + \lceil \log k \rceil$ bits and be valid input to the function $F_s(\cdot)$. Now we just need to show that, if $F_s(x)$ is pseudorandom, then so is $F'_s(x)$.

Proof. To do this, assume that $F'_s(x)$ is not pseudorandom; i.e. that there exists an adversary $A$ that can distinguish between $F'_s(x)$ and a random oracle with non-negligible advantage. Then we construct an adversary $B$ that uses $A$ to distinguish between $F_s(x)$ and a random oracle.

On $A$’s query $x$, $B$ queries its own oracle $k$ times, using $x \circ 0, \ldots, x \circ (k-1)$ as its inputs, where each number is represented in binary using $\lceil \log k \rceil$ bits. $B$ gets back $k$ bits from its oracle, which it then concatenates together and passes along to $A$ in answer to $A$’s oracle query. This means that if $R_B$ is the function that $B$ is querying, then the function that $A$ queries is

$$R_A(x) = R_B(x \circ 0) \circ \cdots \circ R_B(x \circ (k-1)).$$

If $A$ returns 0, then $B$ returns 0; otherwise, $B$ returns 1.

By the construction of $B$, if $R_B$ is equal to $F_s$, then $R_A$ is equal to $F'_s$. On the other hand, if $R_B$ is a random oracle, then each bit of the string returned was selected at random by $R_B$, and so $R_A$ is equivalent to a random oracle. This means that, whenever $B$ sees a random oracle, $A$ will also see a random oracle, and whenever $B$ sees $F_s$, $A$ will see $F'_s$. This means that $B$ is correct whenever $A$ is correct, so $B$ also has a non-negligible probability of distinguishing $F_s$ from random. This means that $F_s$ is not pseudorandom, which is a contradiction. Therefore, $F'_s$ must be a pseudorandom function. \qed

Problem 2: GGM and Prefix-Constrained PRFs

Solution written by Joshua Liebow-Feeser.

a. $G_0(K)$.

b. Define $Constrain(K, \pi)$ as follows. Let $\pi = \pi_1 \pi_2 \ldots \pi_n$, where each $\pi_i \in \{0, 1\}$, and $n = |\pi|$. Output $K_\pi = G_{\pi_n} \ldots G_{\pi_2}(G_{\pi_1}(K)) \ldots$.

c. Define $Eval(K_\pi, x)$ as follows. Let $x = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1\}$. Recall that $n < k$ is the length of the prefix $\pi$. If $x_1 \ldots x_n \neq \pi$, fail. Otherwise, output

$$G_{x_k} \ldots G_{x_{n+2}}(G_{x_{n+1}}(K_\pi)) \cdots.$$
This is correct because $K_\pi = G_{\pi_n}(\ldots G_{\pi_2}(G_{\pi_1}(K))\ldots)$ and $\pi$ is a prefix of $x$, so
\[
G_{\pi_k}(\ldots G_{x_{n+2}}(G_{x_{n+1}}(K_\pi))\ldots) = G_{x_k}(\ldots G_{x_{n+2}}(G_{x_{n+1}}(\ldots G_{\pi_2}(G_{\pi_1}(K))\ldots))\ldots) = F(K, x_1x_2\ldots x_k).
\]

d. **Proof by contradiction.** Assume that this construction is not spring-break-secure, and thus that there exists a PPT algorithm, $A$, capable of distinguishing between the two experiments with non-negligible probability.

Given this, we want to construct $B$, an algorithm capable of breaking GGM, as follows. Let $B$ be given oracle access to $O_B$, which is either $F(s, \cdot)$ for $s \leftarrow \{0,1\}^k$, or a random oracle. $B$ runs $A$ until $A$ chooses a prefix $\pi$, at which point $B$ chooses $K_\pi \leftarrow \{0,1\}^k$, and gives it to $A$.

On queries $x \in X_\pi$ from $A$, $B$ queries its own oracle and responds with $O_B(x)$. On any queries $x \not\in X_\pi$, $B$ fails. $B$ outputs what $A$ outputs.

To begin our analysis, we need a simple lemma.

**Lemma 1.** Given a fixed $\pi$ and $K$ ($|\pi| \le k$), the following two distributions are indistinguishable:
\[
D_0 = \{s \leftarrow \{0,1\}^k : s\} \\
D_1 = \{s \leftarrow \{0,1\}^k : \text{Constrain}(s, K)\}
\]

**Proof.** First, we observe that the following three distributions are indistinguishable from one another:
\[
D'_0 = \{s \leftarrow \{0,1\}^k : G_0(s)\} \\
D'_1 = \{s \leftarrow \{0,1\}^k : s\} \\
D'_2 = \{s \leftarrow \{0,1\}^k : G_1(s)\},
\]

where $G_0$ and $G_1$ apply $G$ to the seed and then return the first or second half of the output, respectively. Since $G$ is a PRG, its output is indistinguishable from random. Thus, half of its output is also indistinguishable from a random string of the same length (otherwise we could distinguish $G$ from random by only looking at half of its output). Since $D'_0 \approx D'_1$ and $D'_1 \approx D'_2$, by transitivity of indistinguishability we have $D'_0 \approx D'_2$.

**Constrain** is simply a sequence of applications of $G_0$ or $G_1$ with a randomly generated seed. By our previous argument, the intermediate state after only one application of $G_0$ or $G_1$ is indistinguishable from random. Thus, the next application of $G_0$ or $G_1$ is applied to a seed which is indistinguishable from random, so its output is also indistinguishable from random. This continues for the rest of the $G_0$s and $G_1$s until the output, $K_\pi$, which is thus also indistinguishable from random. (Note that the number of applications of $G_0$ or $G_1$ is polynomial in $k$.)

Given this lemma, we are ready to proceed.

Consider the case in which $B$'s input is a real instance of GGM. Then $B$ will respond to $A$'s queries exactly like the constrained PRF in the wild: $B$ only responds to queries when $x \in X_\pi$, and when it does respond, it responds using $O_B$, which is, in these cases, equivalent to $Eval(\text{Constrain}(K, \pi), x)$ (although $K$ is not actually known to $B$). Importantly, the value of $K_\pi$ given to $A$ (i.e. a random string) is, as shown in Lemma 1, indistinguishable from a legitimate output of $\text{Constrain}(K, \pi)$. Thus, $A$ cannot distinguish between its current interaction and Exp 1 as defined in the homework prompt. Thus, $B$ is correct in this case with exactly the same probability that $A$ is in Exp 1.
Consider now the case in which \( B \) is accessing a random oracle. Then \( B \) is also acting like a random oracle to \( A \) (albeit one constrained to only accept inputs from \( X_\pi \)). Further, as we showed above, the value \( K_\pi \) that is given to \( A \) is indistinguishable from the output of \( \text{Constrain}(K, \pi) \). Thus, \( A \)'s inputs in this case are indistinguishable from the input it would get if it were actually run in Exp 2 as defined in the homework prompt. Thus, \( B \) will guess correctly in this case with exactly the same probability that \( A \) does in Exp 2.

Thus, \( B \) behaves with exactly the same probabilistic behavior as \( A \), and since \( A \) can distinguish Exp 1 from Exp 2, \( B \) can distinguish between its two experiments (Exp 1 and Exp 2 as defined in lecture 13), and thus can break GGM, which is a contradiction. Thus, our construction must be spring-break-secure.

**Problem 3: PRFs/PRPs as MACs**

a. *Proof.* For the sake of contradiction, assume that it is not a MAC. Let us define a PPT \( A \) which uses Eve to determine whether an oracle \( O \) is a PRF or a random function \( R \).

First, \( A \) invokes \( \text{Eve}_k \). Whenever \( \text{Eve}_k \) puts a query \( m_i \) onto its query tape, provide \( \text{Eve}_k \) with the output of \( O \) on \( m_i \) (i.e. \( O(m_i) \)). After polynomially many queries, \( \text{Eve}_k \) should produce an output \((m, x) \notin Q \). (If it breaks down without an output, or runs longer than it is supposed to, or if \((m, x) \in Q \), then we just set \((m, x) \) at random from the set of \( k \)-bit strings.) Now, \( A \) calculates \( O(m) \), and compares it to \( x \). If \( O(m) = x \), then \( A \) outputs “pseudorandom”; otherwise, it outputs “random”.

Note that, in the case where \( O \) is actually \( F_s \) for some \( s \), \( A \) will be correct whenever \( \text{Eve}_k \) successfully forged a MAC, i.e. with probability \( \epsilon(k) \). Conversely, whenever \( O \) is a random function, \( A \) is correct unless \( \text{Eve}_k \) manages to guess \( R(m) \) correctly. Since \( R(m) \) is completely random, and independent of any observed values, this happens with probability exactly \( 2^{-k} \), so \( A \) will be correct with probability \( 1 - 2^{-k} \).

\[
\Pr[A \text{ is correct}] = \frac{1}{2} \Pr[A \text{ is correct} \mid O \text{ is random}] + \frac{1}{2} \Pr[A \text{ is correct} \mid O \text{ is a PRF}] \\
= \frac{\epsilon(k)}{2} + \frac{1 - 2^{-k}}{2}
\]

Thus, \( A \) will have an advantage of \( \epsilon(k) - \nu(k) \) for negligible \( \nu \), which is non-negligible. This contradicts our assumption that \( F_s \) is a PRF, so our assumption must be false: \( F_s \) is a MAC.

b. A MAC is not always a PRF.

*Proof.* Consider a MAC \( M_s \), and let \( M'_s(m) = 0 \circ M_s(m) \) for all \( m \) and for all \( s \). \( M'_s \) is still a MAC by a simple reduction (i.e. an adversary could use a forged \((m, M'_s(m))\) to forge an equivalent \((m, M_s(m))\) pair). But \( M'_s \) cannot be a PRF, as we can distinguish it from random with high probability.

We can define a distinguisher \( A \) which, given an oracle \( O \) that is either \( M'_s \) or random, samples a random value \( x \) and calculates \( O(x) \). If the first bit of the output is 0, then we guess “pseudorandom”; otherwise, we guess “random”. Since half of all random outputs start with 0, but all outputs of \( M'_s \) begin with 0, \( A \) will be correct with probability 3/4.

c. Yes, if MACs exist then PRFs must also exist. We asked for a brief justification, but we provide a complete proof for reference.
Proof. Recall that, by HILL, the existence of OWFs implies the existence of PRGs. Then, by GGM, the existence of PRGs implies the existence of PRFs. Thus, it is sufficient to show that MAC ⇒ OWF.

Let \( \{M_s : \{0, 1\}^{|s|} \rightarrow \{0, 1\}^{p(|s|)}\} \) be a MAC. Let us define a function \( f : \{0, 1\}^{k+2k^2} \rightarrow \{0, 1\}^{2kp(k)} \), where\(^1\)
\[
f(x) = f(s, m_1, \ldots, m_{2k}) = m_1 \circ \cdots \circ m_{2k} \circ M_s(m_1) \circ \cdots \circ M_s(m_{2k}).
\]

We will now show that inverting this function is equivalent to forging authentication.

Assume for the sake of contradiction that there exists some adversary \( A_k \) that inverts \( f \). We will construct \( Eve_k \), that can output a forged \( (m, M_s(m)) \) with non-negligible probability.

Let us pause for a moment, and note that this problem may be harder than you thought at first. Just because we have inverted the function, obtaining a possible shared secret \( s' \), does not mean that it is the same secret \( s \) that Alice and Bob have! Care must be taken to ensure that \( s' \) is helpful to Eve, even if it is not equal to \( s \).

First, our PPT \( Eve_k \) samples random messages \( m_1, \ldots, m_{2k} \), and queries \( M_s \) on each one, obtaining \( x_1, \ldots, x_{2k} \). Then, \( Eve_k \) runs \( A_k(x_1, \ldots, x_{2k}) \), obtaining \( (s', m_1, \ldots, m_{2k}) \). It then chooses a random \( m \), finally outputting \( (m, M_{s'}(m)) \). Our claim is that, with non-negligible probability, \( M_s(m) = M_{s'}(m) \).

Suppose that \( s \) is fixed. Let \( bad(s) \) denote the strings \( s' \in \{0, 1\}^{|s|} \) such that
\[
Pr[m \leftarrow \{0, 1\}^{|s|} : M_s(m) = M_{s'}(m)] \leq \frac{1}{2}.
\]

If the \( s' \) given to us by \( A_k \) is not in \( bad(s) \), then our \( Eve_k \) succeeds with probability at least \( \frac{1}{2} \). Thus, we want to make sure that there are no bad \( s' \)'s among the preimages of \( f(s, m_1, \ldots, m_{2k}) \).

First, by the union bound,\(^2\) we have that
\[
p_{bad} = Pr[s \leftarrow \{0, 1\}^k; \{m_i \leftarrow \{0, 1\}^k : 1 \leq i \leq 2k\} : \exists s' \in f^{-1}(f(\ldots)) \cap bad(s)]
\]
\[
\leq \sum_{s \in \{0, 1\}^k} Pr[s] \sum_{\{m_i \leftarrow \{0, 1\}^k : 1 \leq i \leq 2k\} : \exists s' \in f^{-1}(f(\ldots)) \cap bad(s)} Pr[\{m_i \leftarrow \{0, 1\}^k : 1 \leq i \leq 2k\} : s' \in f^{-1}(f(\ldots))].
\]

We can then simplify this expression further to find a more explicit bound on \( p_{bad} \):
\[
p_{bad} \leq \sum_{s \in \{0, 1\}^k} Pr[s] \sum_{s' \in bad(s)} Pr[\{m_i \leftarrow \{0, 1\}^k : 1 \leq i \leq 2k\} : M_{s'}(m_i) = M_s(m_i) \forall 1 \leq i \leq 2k]
\]
\[
\leq \sum_{s \in \{0, 1\}^k} Pr[s] \sum_{s' \in bad(s)} 2^{-2k}
\]
\[
< \sum_{s \in \{0, 1\}^k} Pr[s] 2^k 2^{-2k}
\]
\[
< 2^{-k}.
\]

Thus, the probability that there exists a bad \( s' \) is negligible. Assuming \( A_k \) succeeds with probability \( \epsilon(k) \), this leaves \( Eve_k \) with probability \( \frac{\epsilon(k)(1-2^{-k})}{2} \) of success. This contradicts the assumption that \( M_s \) was a MAC, and thus \( f \) must be one-way.

\(^1\)This function doesn’t work on every input size as stated, but we can define a function \( f' \) from any input size \( j \) which finds the maximum \( k \) such that \( k + 2k^2 \leq j \) and runs \( f \) on the first \( k + 2k^2 \) bits of its input.

\(^2\)\( Pr[A \cup B] \leq Pr[A] + Pr[B] \)
One common erroneous solution is to go directly from MAC to PRF. This is very difficult. The problem is that GL bit, i.e., taking a random \( r \) and setting \( F_s(x) = M_s(x).r \) won’t work. This is because a MAC can be designed such that \( M_s(x).r \) is always 0! So there is hope only if \( r \) is kept secret. The actual proof is tricky.

Problem 4: Collision-resistant hash functions

a. Proof. As we have seen before, it suffices to show that we can break RSA for \( y \in \mathbb{QR}_n \).

We note that a collision (for some \( n \) and \( y \)) means that we have an \( x \neq x' \) such that
\[
H_{(n,y)}(x) = y^x = y^{x'} = H_{(n,y)}(x').
\]
In particular, we have \( x \neq x' \) such that \( y^x = y^{x'} \), which means that
\[
y^x - x' \equiv 1 \pmod{n}, \text{ and } x - x' \equiv 0 \pmod{|y|},
\]
where \( |y| \) denotes the order of \( y \). Therefore, \( x - x' = k|y| \) for \( k \in \mathbb{Z} \), and since we know that \( x \neq x' \) we can be sure that \( k \neq 0 \).

To break RSA, we need to find (for a given \( e \)) a \( d \) such that
\[
ed \equiv 1 \pmod{|y|},
\]
or \( ed = 1 + \ell|y| \) for \( \ell \in \mathbb{Z} \). If we had \( |y| \) we could just perform the extended Euclidean algorithm using \( e \) and \( |y| \) to find such a value. But, all we have is \( k|y| \). Since \( e \) and \( k \) may not be relatively prime, we divide by \( \gcd(e, k|y|) \) which, since \( \gcd(e, |y|) = 1 \), is \( \gcd(e, k) \). Now, we perform the extended Euclidean algorithm using \( e \) and \( k|y| \) \( \gcd(e, k|y|) \) to get a \( d \) such that
\[
1 = ed + \ell \cdot \left( \frac{k|y|}{\gcd(e, k|y|)} \right).
\]
If we call this gcd value \( m \), we can rewrite the above equation as: \( 1 = ed + \ell \cdot \frac{k}{m}|y| \). Now, recall that \( m = \gcd(e, k) \), which means that \( m \mid k \) and therefore that \( \frac{k}{m} \) is an integer. Therefore, \( \frac{ed}{m} \) is also an integer, and we have found a \( d \) such that \( ed = 1 - j|y| \) for \( j \in \mathbb{Z} \), so \( ed \equiv 1 \pmod{|y|} \).

Thus,
\[
y^{ed} \equiv y^{(1-j|y|)} \pmod{n} \equiv y \pmod{n},
\]
and we have broken RSA. Since we believe breaking RSA to be hard, we have that this hash function must be collision resistant. \( \square \)

Problem 5: Merkle Trees

a. The domain of \( Merkle_{pk,n} \) is \( \{(0,1)^*\}^n \). We intended to ask for the codomain, which is \( \{0,1\}^k \).

To show that \( (G, Merkle_{(\cdot)_n}) \) is a family of collision-resistant hash functions, we use a reduction. So, we assume that there exists an adversary \( \mathcal{A} \) such that
\[
\Pr[pk \leftarrow G(1^k); ((x_0, \ldots, x_{n-1}), (y_0, \ldots, y_{n-1})) \leftarrow \mathcal{A}(pk) : x_i \neq y_i \text{ for some } 0 \leq i < n \wedge Merkle_{pk,n}(x_0, \ldots, x_{n-1}) = Merkle_{pk,n}(y_0, \ldots, y_{n-1})] = \epsilon(k)
\]

HW 8 Solutions-5
for some non-negligible $\epsilon(\cdot)$, and we use this $A$ to construct a $B$ that breaks the collision-resistance of $(G, H_{(\cdot)})$; that is, a $B$ such that

$$\Pr[pk \leftarrow G(1^k); (x_1, x_2) \leftarrow B(pk) : x_1 \neq x_2 \land H_{pk}(x_1) = H_{pk}(x_2)] = \epsilon'(k)$$

for some non-negligible $\epsilon'(\cdot)$.

On input $pk$, $B$ will run $A(pk)$ to obtain values $(x_0, \ldots, x_{n-1})$ and $(y_0, \ldots, y_{n-1})$. $B$ will then iteratively compute the labels in the Merkle trees and try to find a collision. More formally, for all $0 \leq i \leq n-1$, $B$ computes $u_i = H_{pk}(x_i)$ and $v_i = H_{pk}(y_i)$.

If it is the case that $u_i = v_i$ but $x_i \neq y_i$, $B$ outputs the pair $(x_i, y_i)$. If not, it continues until it has created the labels for all the leaves. If $B$ does not find a collision in this process, it now iterates through all $\ell - 1 \geq j \geq 0$ (to construct the interior of the trees). At each step, it iterates for all $0 \leq k \leq 2^j - 1$ and computes $u_k = H_{pk}(u_{k0} \circ u_{k1})$ and $v_k = H_{pk}(v_{k0} \circ v_{k1})$. If there is some $k$ such that $u_k = v_k$ but $u_{k0} \circ u_{k1} \neq v_{k0} \circ v_{k1}$, we have found a collision and so we output $(u_{k0} \circ u_{k1}, v_{k0} \circ v_{k1})$.

Now, let’s analyze the success probability of this algorithm $B$. Since $A$’s input $pk$ is output by the generation algorithm $G(\cdot)$, it is distributed as expected and therefore $A$ will succeed with the same probability $\epsilon(k)$. We know that in the output of $A$, there exists some $i$ such that $x_i \neq y_i$, but $u_e = v_e$, where $u$ and $v$ are defined as above (here just the roots of the Merkle trees). There are essentially two possibilities: either $H_{pk}(x_i) = H_{pk}(y_i)$ or not. In the first case, it is clear that $B$ succeeds, as it will find this collision in its first iterative process. In the second case, there must be some string $s$ such that $u_s = v_s$ but $u_{s0} \circ u_{s1} \neq v_{s0} \circ v_{s1}$. Since $u_s = H_{pk}(u_{s0} \circ u_{s1})$ and $v_s = H_{pk}(v_{s0} \circ v_{s1})$, this is a collision and $B$ will find it in its second iterative process. Therefore, $B$ will succeed whenever $A$ succeeds, which means that $B$ succeeds with non-negligible probability $\epsilon(k)$.

b. To show this, we again use a reduction. So, we assume there exists an adversary $A$ that breaks the soundness of $M_{\text{Verify}}$; i.e. such that

$$\Pr[pk \leftarrow G(1^k); (v, x, x', i, a, a') \leftarrow A(pk) : x \neq x'$$

$$\land M_{\text{Verify}}_{pk,n}(v, x, i, a) \land M_{\text{Verify}}_{pk,n}(v, x', i, a')] = \epsilon(k)$$

for some non-negligible $\epsilon(\cdot)$, and we want to construct an adversary $B$ that breaks the collision resistance of $(G, H_{(\cdot)})$, so that

$$\Pr[pk \leftarrow G(1^k); (x_1, x_2) \leftarrow B(pk) : x_1 \neq x_2 \land H_{pk}(x_1) = H_{pk}(x_2)] = \epsilon'(k)$$

for some non-negligible $\epsilon'(\cdot)$.

On input $pk$, $B$ will run $A$ on input $pk$ to obtain a tuple $(v, x, x', i, a, a')$, where $a = (u_{t_1}, \ldots, u_{t_k})$ and $a' = (v_{t_1}, \ldots, v_{t_k})$. As in part a., we need to try to find the collision by iterating through the nodes in the tree. So, we first compute $u_t = H_{pk}(x)$ and $v_t = H_{pk}(x')$. If $u_t = v_t$, output $(x, x')$. Otherwise, for all values $\ell - 1 \geq j \geq 0$, let $I_j$ be the $j$-th bit prefix of the binary string $i$ (as in the problem statement). If the $j + 1$-st bit of $i$ is 0, let

$$u_{I_j00} = u_{I_{j+1}}$$
$$v_{I_j00} = v_{I_{j+1}}$$
$$u_{I_j01} = u_{I_{j+1}}$$
$$v_{I_j01} = v_{I_{j+1}}.$$
Otherwise, if it is 1, flip these assignments; i.e. let

\begin{align*}
  u_{I,j_0} &= u_{I,j_1} \\
  v_{I,j_0} &= v_{I,j_1} \\
  u_{I,j_0} &= u_{I,j_1+1} \\
  v_{I,j_0} &= v_{I,j_1+1}.
\end{align*}

Now, let

\begin{align*}
  u_I &= H_{pk}(u_{I,j_0} \circ u_{I,j_1}), \text{ and} \\
  v_I &= H_{pk}(v_{I,j_0} \circ v_{I,j_1}).
\end{align*}

If \( u_I = v_I \) but \( u_{I,j_0} \circ u_{I,j_1} \neq v_{I,j_0} \circ v_{I,j_1} \), output \((u_{I,j_0} \circ u_{I,j_1}, v_{I,j_0} \circ v_{I,j_1})\).

Now we just need to analyze \( B \)'s success probability. As in part a., the input to \( A \) will be correct and so \( A \) will succeed with probability \( \epsilon(k) \). Now let’s suppose that \( A \) succeeds. Then \( A \) outputs a tuple such that \( x \neq x' \), and both \( M_{\text{Verify}}_{pk,n}(v, x, i, a) \) and \( M_{\text{Verify}}_{pk,n}(v, x', i, a') \) are true. Then \( u_e = v_e \), and there are two possible cases: either \( u_i = v_i \), or \( u_i \neq v_i \). In the first case, \( B \) immediately succeeds. In the second case, we know (as in part a.) that there must exist some interior location, where for some \( 0 \leq j \leq i - 1 \) it is the case that \( u_I = v_I \) but \( u_{I,j_0} \circ u_{I,j_1} \neq v_{I,j_0} \circ v_{I,j_1} \). \( B \) will find these values, meaning \( B \) succeeds whenever \( A \) succeeds. Therefore \( B \) succeeds with the same non-negligible probability \( \epsilon(k) \).