I \( \Sigma \)-Protocol for \( \mathcal{P} \)

We consider an alphabet \( Z \), a polynomial \( P \), and a predicate \( R \). We assume that \( R \) can be computed in polynomial time. Given \( x \in Z^* \), we let

\[
R_x = \{ w \in Z^*: R(x, w) \text{ and } |w| \leq P(|x|) \}
\]

where \( |x| \) denotes the length of \( x \). We define the language \( L \) from \( R \) by

\[
L = \{ x \in Z^*: R_x \neq \emptyset \}
\]

Q. In this question, we assume that there is an algorithm \( \mathcal{A} \) such that for any \( x \in L \), we obtain \( \mathcal{A}(x) \in R_x \) and that for any \( x \in Z^* \), the running time of \( \mathcal{A}(x) \) is bounded by \( P(|x|) \).

Construct a \( \Sigma \)-protocol for \( L \). Carefully specify all protocol elements and prove all properties which must be satisfied.

Let \( \varepsilon \) be a word of length 0.

- We define \( \mathcal{P}(x,w) = \varepsilon \) and \( \mathcal{P}(x,w,e) = \varepsilon \).
- We take the set of challenges \( E = \{ \varepsilon \} \). We could actually take any set of challenges with polynomially bounded length.
- The verification algorithm \( V(x,a,e,z) \) first computes \( w = \mathcal{A}(x) \), then checks if \( R(x,w) \) holds.
- Clearly, this protocol satisfies completeness (\( x \in L \) is accepted by the verifier when the protocol is honestly run).
- Clearly, the algorithms run in polynomial time in terms of \( |x| \).
- To define a polynomial time extractor based on some values \( x,a,e,e',z,z' \) such that \( V(x,a,e,z) \) and \( V(x,a,e',z') \) hold, and \( e \neq e' \), we simply compute \( w = \mathcal{A}(x) \). Clearly, we obtain a polynomial-time extractor.
- To define a simulator \( S(x,e) \), we just take \( (a,z) = (e,e) \). Clearly,

\[
\Pr[S(x,e) = (a,z)] = \Pr[\mathcal{P}(x,w) = a, \mathcal{P}(x,w,e) = z]
\]

So, we obtain a polynomial-time simulator.

So, all properties of a \( \Sigma \)-protocol are satisfied.
II OR Proof


Let $Z = \{0, 1\}$ be an alphabet. We consider two $\Sigma$-protocols $\Sigma_1$ and $\Sigma_2$ for two languages $L_1$ and $L_2$ over the alphabet $Z$ defined by two predicates $R_1$ and $R_2$. We assume that $\Sigma_1$ and $\Sigma_2$ use the same challenge set $E$ which is given a group structure with a law $\cdot$. For $\Sigma_i, i \in \{1, 2\}$, we denote $P_i$ the prover algorithm, $V_i$ the verification predicate, $E_i$ the extractor, and $S_i$ the simulator.

Q.1 (AND proof) Construct a $\Sigma$ protocol $\Sigma = \Sigma_1 \text{ AND } \Sigma_2$ for the language defined by

$$R((x_1, x_2), (w_1, w_2)) \iff R_1(x_1, w_1) \text{ AND } R_2(x_2, w_2)$$

The prover and the verifier are simply defined by a parallel execution of $\Sigma_1$ and $\Sigma_2$ together with the same challenge. So are the extractor and the simulator.

More precisely, $P((x_1, x_2), (w_1, w_2); r_1, r_2)$ runs $P_i(x_i, w_i; r_i) = a_i$ for $i = 1, 2$ and yield $(a_1, a_2)$. Upon challenge $e \in E$, $P((x_1, x_2), (w_1, w_2), e; r_1, r_2)$ runs $P_i(x_i, w_i, e; r_i) = z_i$ for $i = 1, 2$ and yield $(z_1, z_2)$. The verification holds $V((x_1, x_2), (a_1, a_2), e, (z_1, z_2))$ if and only if both $V_i(x_i, a_i, e, z_i)$ hold for $i = 1, 2$. The extractor $E((x_1, x_2), (a_1, a_2), e, e', (z_1, z_2), (z_1', z_2'))$ runs $w_i = E_i(x_i, a_i, e, e', z_i, z_i')$ for $i = 1, 2$ and yield $(w_1, w_2)$. The simulator $S((x_1, x_2), e)$ runs $(a_1, z_1) = S_i(x_i, e)$ for $i = 1, 2$ and yields $((a_1, a_2), (z_1, z_2))$.

Note: it is important to use the same challenge for both protocols in order to avoid troubles in the extraction.

(OR proof) In the remaining of the exercise, we now let

$$R((x_1, x_2), w) \iff R_1(x_1, w) \text{ OR } R_2(x_2, w)$$

This predicate defines a new language $L$. We construct a new $\Sigma$-protocol $\Sigma = \Sigma_1 \text{ OR } \Sigma_2$ for $L$ by

- $P((x_1, x_2), w; r_1, r_2)$ finds out $i$ such that $R_i(x_i, w)$ holds, sets $j = 3 - i$, then picks a random $e_j \in E$ and runs $S_j(x_j, e_j; r_1) = (a_j, e_j, z_j)$. Then, it runs $P(x_i, w; r_2) = a_i$ and yield $(a_1, a_2)$.
- Upon receiving $e$, $P((x_1, x_2), w; e; r_1, r_2)$ sets $e_i = e - e_j$, runs $P(x_i, w, e_i; r_2) = z_i$ and yields $(e_1, e_2, z_1, z_2)$.

The verification predicate is

$$V((x_1, x_2), (a_1, a_2), e, (e_1, e_2, z_1, z_2)) \iff \begin{cases} e = e_1 + e_2 \text{ AND } \\ V_1(x_1, a_1, e_1, z_1) \text{ AND } \\ V_2(x_2, a_2, e_2, z_2) \end{cases}$$

Q.2 Show that $\Sigma$ is complete and works in polynomial time.
The protocol $P$ is a finite sequence of polynomial time operations or subroutines, so it is polynomial. Since $V_1$ and $V_2$ have a polynomially bounded complexity, so does $V$. We already know that $E$ is polynomially samplable. So $\Sigma$ works in polynomial time (except that we did not specify yet the extractor and the simulator).

If the protocols are honestly run, we have $S_j(x_j, e_j) \rightarrow (a_j, e_j, z_j)$. So, by the property of the simulator for $\Sigma_j$, we have that $V_j(x_j, a_j, e_j, z_j)$ holds. Since $w$ is a correct witness for $x_i$ in $\Sigma_i$, since $P(x_i, w; r_2) = a_i$ and $P(x_i, w, e_i; r_2) = z_i$, due to the completeness of $\Sigma_i$, we have that $V_i(x_i, a_i, e_i, z_i)$ holds. Since we further have $e_i = e - e_j$, the last condition for $V((x_1, x_2), (a_1, a_2), e, (e_1, e_2, z_1, z_2))$ to hold is satisfied. So, $\Sigma$ satisfies the completeness property of $\Sigma$-protocols.

Q.3 Construct an extractor $E$ for $\Sigma$ and show that it works, in polynomial time.

If $V((x_1, x_2), (a_1, a_2), e, (e_1, e_2, z_1, z_2))$ and $V((x_1, x_2), (a_1, a_2), e', (e_1', e_2', z_1', z_2'))$ hold with $e \neq e'$, we must have either $e_1 \neq e_1'$ or $e_2 \neq e_2'$. Let assume that $e_1 \neq e_1'$. Then, we know that $V_1(x_1, a_1, e_1, z_1)$ and $V_1(x_1, a_1, e_1', z_1')$ hold. So, we can run the $E_1$ extractor on $(x_1, a_1, e_1, e_1', z_1, z_1')$ to extract a witness $w$ for $x_1$ in $L$. Clearly, $w$ is also a witness for $(x_1, x_2)$ in $L$. The method is similar in the case $e_2 \neq e_2'$.

Clearly, we obtain a polynomially bounded extractor.

Q.4 Construct a simulator $S$ for $\Sigma$ and show that it works, in polynomial time.

Given $(x_1, x_2)$ and $e$, we pick a random $e_1$ and let $e_2 = e - e_1$. Then, we run $S_1(x_1, e_1) \rightarrow (a_1, e_1, z_1)$ and $S_2(x_2, e_2) \rightarrow (a_2, e_2, z_2)$. The output is $((a_1, a_2), e, (e_1, e_2, z_1, z_2))$. This defines our simulator $S$.

Clearly, this works in polynomial time.

We let $a = (a_1, a_2)$ and $z = (e_1, e_2, z_1, z_2)$. We have

$$\Pr[S \rightarrow a, e, z|e] = \sum_{e_1 + e_2 = e} \Pr[e_1] \Pr[S_1 \rightarrow a_1, e_1, z_1|e_1] \Pr[S_2 \rightarrow a_2, e_2, z_2|e_2]$$

Since $S_1$ and $S_2$ are simulators for $\Sigma_1$ and $\Sigma_2$, we have

$$\Pr[S \rightarrow a, e, z|e] = \sum_{e_1 + e_2 = e} \Pr[e_1] \Pr[S_j \rightarrow a_j, e_j, z_j|e_j] \Pr[S_i \rightarrow a_i, e_i, z_i|e_i]$$

for whatever pair $(i, j)$ such that $\{i, j\} = \{1, 2\}$. We let $i$ be random defined by $P$. Clearly, the above sum equals $\Pr[S \rightarrow a, e, z|e]$. So, $S$ satisfies the property of a simulator for $\Sigma$. 
III Smashing SQUASH-0

The exercise is inspired by Smashing SQUASH-0 by Ouafi and Vaudenay. Published in the proceedings of Eurocrypt’09 pp. 300–312, LNCS vol. 5479, Springer 2009.

We consider an access control protocol called SQUASH-0 in which a client and a server hold a secret key $K$. In the protocol, the server sends a challenge $C$. The client must respond with

$$S = (\text{stoi}(C \oplus K))^2 \mod N$$

for a given modulus $N$, where $\text{stoi}$ is a function transforming a bitstring into an integer by $\text{stoi}() = 0$ for the zero-length bitstring $e$, and

$$\text{stoi}(b||s) = b + 2 \times \text{stoi}(s)$$

for any bit $b \in \{0, 1\}$ and any bitstring $s$. By convention, the least significant bit has position 0. We further assume that $N$ is larger than $K$ and $C$.

Q.1 Let $c_i$ be $-1$ raised to the power of the bit position $i$ in $C$. Let $k_i$ be $-1$ raised to the power of the bit position $i$ in $K$.

Show that

$$S = \left( \frac{1}{4} \sum_{i,j} 2^{i+j} c_i c_j k_i k_j - \frac{2^\ell - 1}{2} \sum_i 2^i c_i k_i + \frac{(2^\ell - 1)^2}{4} \right) \mod N$$

where $\ell$ is the bitlength of $N$.

The XOR of two bits in the $\pm 1$ representation is obtained by a regular multiplication. The $\pm 1$ representation of bits can be converted to a $0$-$1$ representation by $x \mapsto \frac{1-x}{2}$. So,

$$\text{stoi}(C \oplus K) = \sum_i 2^i \frac{1 - c_i k_i}{2} = \frac{2^\ell - 1}{2} - \frac{1}{2} \sum_i 2^i c_i k_i$$

By squaring it we obtain the result for $S$.

The SQUASH-0 proposal suggests to use Mersenne numbers for $N$. incidentally, we obtain $2^\ell - 1 = N$. We deduce

$$S = \left( \frac{1}{4} \sum_{i,j} 2^{i+j} c_i c_j k_i k_j \right) \mod N$$

In what follows, we assume that $N = 2^\ell - 1$. Deduce

$$S = \left( \frac{1}{4} \sum_{i,j} 2^{i+j} c_i c_j k_i k_j \right) \mod N$$

Q.2 Deduce that by using about $\ell^2$ challenges and their responses, an adversary could recover $K$ by solving a linear system of $O(\ell^2)$ equations with $\frac{\ell(\ell-1)}{2}$ unknowns.

As an example, consider $\ell = 1024$. What is the complexity of the attack?

Hint: define $\kappa_{i,j} = k_i k_j$. 
We let $k_{i,j} = k_j$ for $i < j$. For $i = j$, we have $k_i k_j = 1$. For $i > j$, we have $k_i k_j = k_{j,i}$. So, all $k_i k_j$ can be expressed in terms of $k$’s. This way, the equation becomes linear. We have $\frac{t(t-1)}{2}$ unknowns $k$. So, by collecting enough equations (namely, about $t^2$), we can solve the linear system. The complexity of such algorithm is essentially $O(t^6)$. For $t = 2^{10}$, we need $2^{20}$ known challenges and we reach a complexity of $2^{60}$, which is not practical.

Q.3 Given a function $\phi$ mapping a bitstring of length $d$ to a real number, we define

$$\hat{\phi}(V) = \sum_x (-1)^{x \cdot V} \phi(x)$$

where $\cdot$ denotes the dot product between two bitstrings and the sum goes on all bitstrings $x$ of length $d$. For the function $\phi(x) = (-1)^{x \cdot U}$, show that $\hat{\phi}(V) = 2^d$ if $V = U$ and $\hat{\phi}(V) = 0$ otherwise. We write it $\hat{\phi}(V) = 2^d 1_{V = U}$.

We have

$$\hat{\phi}(V) = \sum_x (-1)^{x \cdot (U \oplus V)}$$

When $U \oplus V \neq 0$, this is zero. When $U = V$, this clearly is $2^d$.

Q.4 In a chosen challenge attack, an adversary creates $d$ challenges $C_1, \ldots, C_d$ and all linear combinations of these challenges. Namely, $C(x_1 \ldots x_d) = x_1 C_1 + \cdots + x_d C_d$. Given a $d$-bit vector $x$, we thus define $C(x)$. We write $x$ as an argument of $S$ and $c_i$ as well so that $S(x)$ is the response to challenge $C(x)$ and $c_i(x)$ is $-1$ raised to the power of the bit position $i$ in $C(x)$. Let $U_i$ be the $d$-bit vector consisting of the bit at position $i$ of $C_1, \ldots, C_d$.

Deduce that

$$\hat{S}(V) = \frac{1}{4} \sum_{i,j} 2^{d+i+j} k_i k_j 1_{V = U_i \oplus U_j}$$

Hint: observe $c_i(x) = (-1)^{x \cdot U_i}$ and use Q.1 then Q.3.

The bit at position $i$ of $C(x)$ is clearly $x \cdot U_i$. So,

$$c_i(x) = (-1)^{x \cdot U_i}$$

We now use Q.1. By the definition of $\hat{S}$, we have

$$\hat{S}(V) = \sum_x (-1)^{x \cdot V} \left( \frac{1}{4} \sum_{i,j} 2^{d+i+j} c_i(x) c_j(x) k_i k_j \right) \mod N$$

We can now use our observation and permute the two sums and obtain

$$\hat{S}(V) = \frac{1}{4} \sum_{i,j} 2^{i+j} k_i k_j \sum_x (-1)^{x \cdot (V \oplus U_i \oplus U_j)}$$

We can then use Q.3.
**Q.5** With the same notations, we assume that the function mapping a non-ordered pair \( \{i, j\} \) with \( i \neq j \) to \( U_i \oplus U_j \) behaves like a random function. We further assume that \( d \) is pretty small. For each \( V \), estimate the number of non-ordered pairs \( \{i, j\} \) with \( i \neq j \) such that \( V = U_i \oplus U_j \).

Deduce that we get \( 2^d \) equations modulo \( N \) with \( \ell(\ell - 1)2^{-d-1} \) unknowns \( \kappa_{i,j} \) on average taking values in \( \{-1, +1\} \).

\[
\text{We have } \frac{\ell(\ell-1)}{2} \text{ non-ordered pairs } \{i, j\} \text{ with } i \neq j. \text{ The vector } U_i \oplus U_j \text{ takes values in a set of } 2^d \text{ elements. So, each } V \text{ has (on average) } \ell(\ell - 1)2^{-d-1} \text{ pairs. Therefore, each equation } \hat{S}(V) \text{ uses this amount of unknowns } \kappa_{i,j} = k_ik_j.
\]

**Q.6** We take \( d = 2 \log_2 \ell \) and solve each equation by exhaustive search. Deduce a chosen-challenge attack to break the algorithm.

How many chosen challenges does it use, asymptotically?

What is its complexity?

With \( d = 2 \log_2 \ell \), each equation has \( \frac{1}{2} \) unknown on average. So, exhaustive search works in constant time. We just solve \( O(\ell^2) \) equations using \( O(\ell^2) \) chosen challenges.

1: pick \( C^1, \ldots, C^d \)
2: for each \( x \), define \( C(x) \) and get \( S(x) \)
3: do an FFT transform on \( S \) to get the table \( \hat{S} \)
4: for each \( V \), make an exhaustive search on the expressed \( \kappa_{i,j} = \pm 1 \) in \( \hat{S}(V) \) to recover the \( \kappa \)'s
5: pick \( k_1 \) at random and infer \( k_i \) from \( \kappa_{1,i} \)

The FFT complexity is \( O(d2^d) \). So, the overall complexity is \( O(\ell^2 \log \ell) \). This is much better than \( O(\ell^6) \).
IV PIF Implies PAF

We consider a function family $F_k$ taking inputs of length $\lambda$, making outputs of length $\lambda$, and where the key $k$ is also of length $\lambda$. We consider the two following games:

**Game PIF($A, 1^\lambda$):**
1: pick some random coins $k$ of length $\lambda$
2: pick $r$
3: run $A(r) \rightarrow x$
4: if $|x| \neq \lambda$, output 0 and stop
5: pick a random bit $b$
6: **if** $b = 0$ **then**
7: compute $y = F_k(x)$
8: **else**
9: pick a random $y$ of $\lambda$ bits
10: **end if**
11: run $A(y; r) \rightarrow b'$
12: output $b \oplus b' \oplus 1$

**Game PAF($A, 1^\lambda$):**
1: pick some random coins $k$ of length $\lambda$
2: pick $r$
3: pick a random $x$ of length $\lambda$
4: compute $y = F_k(x)$
5: run $A(y; r) \rightarrow x'$
6: output $1_{x = x'}$

We say that $F_k$ is PIF-secure (resp. PAF-secure) if for all polynomially bounded $A$, we have that $Pr[\text{PIF}(A, 1^\lambda) = 1] - \frac{1}{2}$ (resp. $Pr[\text{PAF}(A, 1^\lambda) = 1]$) is a negligible function in terms of $\lambda$.

Q. Show that if $F_k$ is PIF-secure, then it is PAF-secure.

**Hint:** based on a PAF-adversary $A$ and some coins $p' = r'||p||b''$, define $A'(p') = x$ picked at random from $r'$ then $A'(y, p') = 1$ if $A(y; p) = x$ and $A'(y, p') = b''$ otherwise. By considering $A'$ as a PIF-adversary, look at the link between $Pr[\text{PIF}(A', 1^\lambda) = 1] - \frac{1}{2}$ and $Pr[\text{PAF}(A, 1^\lambda) = 1]$. 
Consider an adversary $A$ who is polynomially bounded. We want to show that $p = \Pr[\text{PAF}(A, 1^\lambda) = 1]$ is negligible.

For this, we define the adversary $A'$ as follows: we let $r' = r \parallel p \parallel b''$ and $A'(r')$ picks a random $x$ using $r'$. Then, $A'(y, r')$ runs $A(y, p) = x''$. If $x = x''$, it answers 1. Otherwise, it answers by $b''$.

When running the game $\text{PIF}(A', 1^\lambda)$, in the $b = 0$ case, we have $x = x''$ with probability $p$ and $A'$ never answers 0. We have $x \neq x''$ with probability $1 - p$ and $A'$ answers 0 with probability $1/2$. So, $A'$ answers 0 with probability $1 - p$. So,

$$\Pr[\text{PIF}(A', 1^\lambda) = 1 | b = 0] = \frac{1 - p}{2}$$

When $b = 1$, $A(y, p)$ has no information about $x$, so $x$ is independent from $x''$ and we have $\Pr[x = x''] = 2^{-\lambda}$. Thus,

$$\Pr[\text{PIF}(A', 1^\lambda) = 1 | b = 1] = 2^{-\lambda} + \frac{1 - 2^{-\lambda}}{2}$$

Finally, we have

$$\Pr[\text{PIF}(A', 1^\lambda) = 1] = \frac{1}{2} - \frac{1}{2} \left( \frac{1 - p}{2} + 2^{-\lambda} + \frac{1 - 2^{-\lambda}}{2} \right) = \frac{1}{2}$$

$$= \frac{p}{4} + \frac{2^{-\lambda}}{4}$$

Since $F_k$ is $\text{PIF}$-secure, we know that $\Pr[\text{PIF}(A', 1^\lambda) = 1] = \frac{1}{2}$ must be negligible. Thus,

$$-\frac{p}{4} + \frac{2^{-\lambda}}{4}$$

is negligible. Since $\frac{2^{-\lambda}}{4}$ is negligible, we obtain that $\frac{p}{4}$ is negligible. So, $p$ is negligible.