Classes of Information Security, Computer Security, Network Security, Wireless Network Security

List of laboratory exercises

instructor: Nicola Laurenti

Fall 2017-18

Laboratory exercise (1 student)

Diffie-Hellman key agreement over a permutation group

Your aim is to implement and evaluate a Diffie-Hellman key agreement scheme, where the group $\mathcal{G}$ is the set of all the distinct permutations of M objects, equipped with the composition operation (i.e., if π_1, π_2 are any two permutations, their composition $\pi_3 = \pi_1 \circ \pi_2$ is the permutation that yields $\pi_3([x_1, \ldots, x_M]) = \pi_1(\pi_2([x_1, \ldots, x_M]))$ for any collection $x_1, \ldots, x_M$).

1. choose a suitable representation of the permutations in $\mathcal{G}$, so that the complexity of calculating a composition is polynomial in M
2. implement the Diffie-Hellman scheme so that the complexity of the protocol is polynomial in M and the output is a uniformly distributed binary key k
3. design and implement a passive attack to your scheme and evaluate its computational complexity vs success probability tradeoff

You will turn in

- a short report (a few pages) including:
  - a description of your implementations, with adequate justification to your choices
  - a plot of the average required computation time vs M, both for the key agreement scheme and for a successful attack
- the code for your implementations (Matlab/Octave, C, Java, or Python are all fine).

Laboratory exercise (2 students)

Digital signature with low density channel codes

Your aim is to implement and evaluate a digital signature scheme, as proposed in

> M. Baldi, M. Bianchi, F. Chiaraluce, J. Rosenthal, and D. Schipani,
> "Using LDGM Codes and Sparse Syndromes to Achieve Digital Signatures,"
> *2013 International Conference on Post-Quantum Cryptography, PQCrypto*, 23 May 2013, pp. 1–15
> available online at http://arxiv.org/abs/1305.5436

You are therefore asked to:

1. implement the key generation, the signature and the corresponding verification scheme and numerically evaluate their computational complexity
2. implement the forging attack described in Section 4.1 of the above paper, and numerically evaluate, for small code sizes, its computational complexity and success probability

You will turn in, through the assignment Moodle page

- a short report (a few pages) including:
  - a description of your implementations, with adequate justifications to your choices
  - the following performance plots
    - a plot of the average required computation time vs the signature length, both for the signature and the verification
    - a plot of the average required computation time vs the private key length, both for the signature and the verification
    - a plot of the average required computation time and success probability vs the signature length for the forging attacks
- the code for your implementations (Matlab/Octave, C, Java, or Python are all fine)

Laboratory exercise (1 student)

Elgamal encryption over a permutation group

Your aim is to implement and evaluate an ElGamal asymmetric encryption scheme, where the group $\mathcal{G}$ is the set of all the distinct permutations of M objects, equipped with the composition operation (i.e., if π_1, π_2 are any two permutations, their composition $\pi_3 = \pi_1 \circ \pi_2$ is the permutation that yields $\pi_3(\,[x_1, \ldots, x_M]\,) = \pi_1(\,\pi_2(\,[x_1, \ldots, x_M]\,)\,)$ for any collection $x_1, \ldots, x_M$).

1. choose a suitable representation of the permutations in $\mathcal{G}$, so that the complexity of calculating a composition is polynomial in M
2. implement the ElGamal scheme so that the complexity of both the encoder and decoder is polynomial in M
3. design and implement a passive attack to your scheme and evaluate its computational complexity vs success probability tradeoff

You will turn in

- a short report (a few pages) including:
  - a description of your implementations, with adequate justification to your choices
  - a plot of the average required computation time vs M, both for the encryption/decryption scheme and for a successful attack
- the code for your implementations (Matlab/Octave, C, Java, or Python are all fine)

Laboratory exercise (2 students)

Information theoretic key agreement example

Your aim is to implement and evaluate an information theoretic key agrement scheme in the channel model with a legitimate transmitter (A), a legitimate receiver (B) and an eavesdropper (E), where:

- the initial randomness is generated by A as a sequence of iid binary symbols with equally likely 0 and 1;
- the probabilistic channel from A to B is a memoryless binary symmetric channel (BSC) with error probability ε_{B}, the probabilistic channel from A to E is a memoryless BSC with error probability ε_{E}, and errors in the two channels are independent;
- the public channel among A,B and E is error-free: both A and B can transmit and/or receive, while E can not transmit but receives everything;
- information reconciliation is performed on each block by making use of a $(7, 4)$ Hamming code;
- privacy amplification is performed on each reconciled block by a *fixed* (and known to all A, B and E) binary Toeplitz matrix of the appropriate size to produce the pair of final binary keys $(k_{\mathrm{A}}, k_{\mathrm{B}})$ of length ℓ .

Carry out your implementation for several different values of the parameters ε_{B}, ε_{E} and ℓ and evaluate numerically:

1. the correctness of the final keys, in terms of the probability $\mathrm{P}\left[k_{\mathrm{A}} \neq k_{\mathrm{B}}\right]$;
2. their uniformity, in terms of the entropies $H(k_{\mathrm{A}})$ and $H(k_{\mathrm{B}})$;
3. their secrecy, in terms of the mutual information $I(k_{\mathrm{A}}; z, c)$ and $I(k_{\mathrm{B}}; z, c)$, where z represents the block received by E over her BSC channel from A, and c is the redundancy transmitted over the public channel for the purpose of information reconciliation

You will turn in, through the assignment Moodle page

- a short report (a few pages) including:
  - a description of your implementation, with adequate justification to your choices;
  - plots of $\mathrm{P}\left[k_{\mathrm{A}} \neq k_{\mathrm{B}}\right]$, $\ell - H(k_{\mathrm{A}})$, $\ell - H(k_{\mathrm{B}})$, $I(k_{\mathrm{A}}; z, c)$, $I(k_{\mathrm{B}}; z, c)$ as a function of the parameters ε_{B}, ε_{E} and ℓ;
  - your final considerations on what values of the parameters yield satisfactory performances, and a comparison between the secret key rate you have obtained and the secrect key capacity of the system.
- the code for your implementation, numerical evaluation and/or simulations (Matlab/Octave, C, Java, or Python are all fine).

Laboratory exercise (1 student)

Information thoretic key agreement over parallel Gaussian channels

Your aim is to implement the optimal power allocation scheme for Information Theoretic Secret Key Agreement over a set of parallel wiretap Gaussian channels with known main and eavesdropper channel gains, and a bound on the maximum transmission power, in a similar fashion to the scheme for Physical Layer Secrecy that is described in the slides *Secrecy rates for multiple channels*. Your algorithm should also compute the achievable secret key rates with the given power bound, and the secret key capacity in the infinite power limit. You are allowed to make use of any programming language of your choice (Matlab/Octave, C, Java, or Python are all fine)

Your algorithm must:

1. take as input the values of the SNR coefficients $\Lambda_{\mathrm{B},i}$ $\Lambda_{\mathrm{E},i}$ and of the maximum total power $P_{\max}$, as a list of numbers in floating point exponential representation (e.g., `1.25e-3` for $1/800 = 0.00125$) separated by spaces in the files `LambdaB.txt`, `LambdaE.txt` and `Pmax.txt` (only one value in this file), respectively
2. compute the amount of power P_i that should be allocated to each subchannel, and the corresponding achievable secret key rates R_i
3. compute the total secret key rate R_{k} and the high-SNR secret key capacity C_{k}
4. output the values of allocated powers and achievable secret key rates as a list of numbers in floating point exponential representation separated by spaces in the files `Powers.txt` and `Rates.txt`, respectively
5. output the values of the total secret key rate and high-SNR secret key capacity as numbers in floating point exponential representation in the files `TotalRate.txt`, `Capacity.txt`, respectively

You will turn in

- a short report (a few pages maximum) including:
  - a description of your implementation, with adequate justification to your choices
  - some example plots of significant cases showing: subchannel gains, power allocation values across the subchannels, achievable secret key rate for each subchannel.
- the code for your implementation

Laboratory exercise (2 students)

Mutual information jamming game over parallel channels

Consider a mutual information jamming game where both the transmitter T and the jammer J can transmit Gaussian zero mean signals on L parallel discrete time Gaussian channels, being limited only by their maximum total transmit powers $P_{\mathsf{T}}, P_{\mathsf{J}}$ respectively. Assume that the noise variance at the receiver is σ^2 on each channel and is known to both T and J, and that all the channel gains $h_{\mathsf{T},i}, h_{\mathsf{J},i}, i = 1, \ldots, L$ are known to both T and J.

Your aim is to implement the optimal transmit power allocation strategies for both T and J, and evaluate the possible results of this game

You are allowed to make use of any programming language of your choice (Matlab/Octave, C, Java, or Python are all fine)

You are therefore asked to:

1. implement an algorithm that, starting from the values of the channel gains $h_{\mathsf{T},i}, h_{\mathsf{J},i}$, $i = 1, \ldots, L$, maximum transmit power P_{T} and noise variance σ^2, and given a jamming power allocation $[p_{\mathsf{J},1}, \ldots, p_{\mathsf{J},L}]$ computes the corresponding *best response* power allocation $[\hat{p}_{\mathsf{T},1}, \ldots, \hat{p}_{\mathsf{T},L}]$ for the transmitter and the resulting mutual information. Your implementation must:
   - take its inputs as a number in scientific (exponential) notation (e.g., `1.25e-3` for $1/800 = 0.00125$) or a list of such numbers separated by spaces in the files `gainsT.txt`, `gainsJ.txt`, `powersJ.txt`, `PmaxT.txt`, `noiseVar.txt`, respectively
   - write its outputs as numbers in scientific notation in the files `bestpowersT.txt`, `bestTmutualInfo.txt`, respectively
2. implement an algorithm that, starting from the values of the channel gains $h_{\mathsf{T},i}, h_{\mathsf{J},i}$, $i = 1, \ldots, L$, maximum transmit power P_{J} and noise variance σ^2, and given a jamming power allocation $[p_{\mathsf{T},1}, \ldots, p_{\mathsf{T},L}]$ computes the corresponding *best response* power allocation $[\hat{p}_{\mathsf{J},1}, \ldots, \hat{p}_{\mathsf{J},L}]$ for the jammer and the resulting mutual information. Your implementation must:
   - take its inputs as a number in scientific notation or a list of such numbers separated by spaces in the files `gainsT.txt`, `gainsJ.txt`, `powersT.txt`, `PmaxJ.txt`, `noiseVar.txt`, respectively
   - write its outputs as numbers in scientific notation in the files `bestpowersJ.txt`, `bestJmutualInfo.txt`, respectively
3. implement an algorithm that, starting from the values of the channel gains $h_{\mathsf{T},i}, h_{\mathsf{J},i}$, $i = 1, \ldots, L$, maximum transmit powers $P_{\mathsf{T}}, P_{\mathsf{J}}$ and noise variance σ^2, computes the *minimax solutions* for T and J, respectively. Does the game have a saddle point solution? Always, never, sometimes? Your implementation must:
   - take its inputs as a number in scientific notation or a list of such numbers separated by spaces in the files `gainsT.txt`, `gainsJ.txt`, `PmaxT.txt`, `PmaxJ.txt`, `noiseVar.txt`, respectively
   - write its outputs as numbers in scientific notation in the files `minimaxTpowers.txt`, `minimaxJpowers.txt`, `minimaxTmutualInfo.txt`, `minimaxJmutualInfo.txt`, respectively
4. simulate the repeated alternating best response game, starting from a random power allocation for the transmitter, and allowing each player at a time to respond to the last choice of his/her adversary. Discuss the long term behaviour of the game. Is there any convergence?

You will turn in

- a short report (a few pages) including:
  - a description of your implementation, with adequate justification to your choices;
  - some example plots of significant cases for best responses, showing: subchannel gains, power allocation values across the subchannels, mutual information for each subchannel;
  - some example plots of significant cases for minimax solutions, showing: subchannel gains, power allocation values across the subchannels, mutual information for each subchannel;
  - some example plots of the time evolution of the mutual information in the repeated alternating best repsonse game
  - the answers to questions in points 3 and 4
- the code for your implementation

Laboratory exercise (1 student)

Naïve arbitrated protocol

A and B want to establish a secure connection based on symmetric cryptography, sharing a secret key k. In order to do that, since each of them has a connection with asymmetric cryptography to C, they want to use the following protocol:

k_{A}	private key of A
k'_{A}	public key of A (known to C)
k_{B}	private key of B
k'_{B}	public key of B (known to C)
k_{C}	private key of C
k'_{C}	public key of C (known to A and B)

1 A : generates nonce $r_{\mathrm{A}} \sim \mathcal{U}(\mathcal{R})$
A $\to$ C : $[\mathrm{id}_{\mathrm{A}}, \mathrm{id}_{\mathrm{B}}, r_{\mathrm{A}}]$

2 C : generates $k \sim \mathcal{U}(\mathcal{K})$, $u_1 = [k, r_{\mathrm{A}}]$, encrypts $x_1 = E_{k'_{\mathrm{A}}}(u_1)$
C $\to$ A : x_1
A : decrypts $u_1 = D_{k_{\mathrm{A}}}(x_1)$

3 A : signs $t_2 = S_k([\mathrm{id}_{\mathrm{A}}, \mathrm{id}_{\mathrm{B}}, r_{\mathrm{A}}])$
A $\to$ B : $x_2 = [\mathrm{id}_{\mathrm{A}}, \mathrm{id}_{\mathrm{B}}, r_{\mathrm{A}}, t_2]$

4 B $\to$ C : $[\mathrm{id}_{\mathrm{A}}, \mathrm{id}_{\mathrm{B}}, r_{\mathrm{A}}]$

5 C : encrypts $x_3 = E_{k'_{\mathrm{B}}}(u_1)$
C $\to$ B : x_3
B : decrypts $u_1 = D_{k_{\mathrm{B}}}(x_3)$

6 B : verifies $b = V_k([\mathrm{id}_{\mathrm{A}}, \mathrm{id}_{\mathrm{B}}, r_{\mathrm{A}}], t_2)$

Your assignment is to:

1) implement the above protocol in your favourite language (Matlab/Octave, C, Java, or Python are all fine), and check its correctness;

2) identifiy its vulnerabilities and devise an attack that exploits them, under reasonable assumptions;

3) implement the attack and evaluate its success probability in dependence of the protocol parameters;

4) suggest improvements to the protocol and implement them.

Provide a description of your solution, with justification of your choices, the code for your implementations, and adequate discussion of the results.

General notes: Implementing your protocol may require you to use several cryptographic primitives, such as symmetric or asymmetric encryption/decryption, signing/verification, one-way functions, cryptographic hash functions, key generation, etc. Feel free to use any reasonable (and correct) implementation of such primitives you find for the language you've chosen, or even simplified models such as a (pseudo-)random oracle. The vulnerability of the protocols should not depend on a poor implementation of such primitives. Also, unlike for cryptographic keys, you should model the random choice of a password as (strongly) non uniform.

Laboratory exercise (1 student)

Naïve message authentication and integrity protection

Your aim is to implement and evaluate the weakness of the following naïve symmetric scheme for message authentication and integrity protection: the message m is a sequence of M bits, the key k is a sequence of K bits, and the authentication tag t is the binary representation of the integer product $s = s_m s_k$, where s_m and s_k are the sum of the digits in the decimal (base 10) representation of m and k, respectively. The tag is transmitted along with the message, and the receiver, which knows k, computes the tag from the received message and checks whether it is identical to the received tag. If so, the message is accepted as authentic.

1. Implement the scheme, and the corresponding verification scheme so that the complexity of both is polynomial in M and K
2. Design and implement a substitution attack to the above scheme that, without knowing the key k, and after blocking and observing a legitimate message/tag pair (x, t) sent by A, replaces it with a different pair $(\tilde{x}, \tilde{t})$ so that $\tilde{x}$ is accepted by B as authentic. Evaluate the computational complexity for this attack
3. Design and implement a forging attack that, without knowing the key k, aims at creating a message/tag pair $(\tilde{x}, \tilde{t})$, such that $\tilde{x}$ is accepted by B as authentic. Evaluate its computational complexity and success probability.

You will turn in, through the assignment Moodle page

- a short report (a few pages) including:
  - a description of your implementations, with adequate justification to your choices;
  - a plot of the average required computation time vs M and K, both for the authentication/verification scheme and for successful attacks;
  - a plot of the success probability vs M and K, for the forging attack;
- the code for your implementation, numerical evaluation and/or simulations (Matlab/Octave, C, Java, or Python are all fine).

Laboratory exercise (1 student)

Naïve Toeplitz authentication

Given $\mathcal{M} = \{0,1\}^n$, $\mathcal{K} = \{0,1\}^{n+m-1}$, $\mathcal{T} = \{0,1\}^m$, let $(S(\cdot,\cdot), V(\cdot,\cdot,\cdot), \mathcal{M}, \mathcal{K}, \mathcal{T})$ be a symmetric authentication / integrity protection mechanism such that

$$t = S(k,r) = T(k)r, \quad t \in \mathcal{T}, r \in \mathcal{M}$$

being $T(k)$ the $m \times n$ Toeplitz matrix identified by k, and the verification function is defined accordingly.

Suppose A wants to authenticate himself to B, with whom he shares the secret key k, by using the following protocol

1 $\mathsf{A} \to \mathsf{B}$: id_A

2 B : generates a nonce $r \sim \mathcal{U}(\mathcal{M})$
$\mathsf{B} \to \mathsf{A}$: r

3 A : calculates $t = S(k,r)$
$\mathsf{A} \to \mathsf{B}$: t

4 B : checks if $V(k,r,t) = \mathsf{ok}$

Your assignment is to:

1) implement the above protocol in your favourite language (Matlab/Octave, C, Java, or Python are all fine), and check its correctness;

2) identifiy its vulnerabilities and devise an attack that exploits them, under reasonable assumptions;

3) implement the attack and evaluate its success probability in dependence of the protocol parameters;

4) suggest improvements to the protocol and implement them.

Provide a description of your solution, with justification of your choices, the code for your implementations, and adequate discussion of the results.

General notes: Implementing your protocol may require you to use several cryptographic primitives, such as symmetric or asymmetric encryption/decryption, signing/verification, one-way functions, cryptographic hash functions, key generation, etc. Feel free to use any reasonable (and correct) implementation of such primitives you find for the language you've chosen, or even simplified models such as a (pseudo-)random oracle. The vulnerability of the protocols should not depend on a poor implementation of such primitives. Also, unlike for cryptographic keys, you should model the random choice of a password as (strongly) non uniform.

Laboratory exercise (1 student)

Non secure random number generation

Your aim is to implement some simple pseudo random number generators (PRNG), and compare them in terms of uniformity and security (i.e., unpredictability by an attacker). Each PRNG outputs a sequence of integers $\{x_0, x_1, \ldots, x_n, \ldots\}$ in the range $\{1, \ldots, p-1\}$ where p is an M-bit prime.

1. Implement the PRNGs defined by the following recursive relationship

$$\begin{aligned} \mathcal{G}_1 &: x_{n+1} = (ax_n + b) \bmod p \quad , \quad a, b, x_0 \in \{1, \ldots, p-1\} \\ \mathcal{G}_2 &: x_{n+1} = (x_n^a + b) \bmod p \quad , \quad a, b, x_0 \in \{1, \ldots, p-1\} \\ \mathcal{G}_3 &: x_{n+1} = (a^{bx_n}) \bmod p \quad , \quad a, b, x_0 \in \{1, \ldots, p-1\} \end{aligned}$$

   so that

   - although M and p are constant (and publicly known) parameters, your implementation does not assume any particular upper limit to their values;
   - the generation of a single value has a computational complexity that is polynomial in M.

2. For each PRNG, and different values of p, evaluate whether p consecutive generated values are uniformly distributed.

3. For each PRNG design and implement a passive attack that aims at predicting the next generated value x_{n+1}, from knowledge of the value of p and observation of the past values $\{x_0, \ldots, x_n\}$, but without knowledge of the values of a and b.

You will turn in

- a short report (a few pages) including:
  - a description of your implementations (both of the PRNGs and the attacks), with adequate justification to your choices
  - a plot of a quantitative description of PRNG uniformity vs M
  - a plot of the average required computation time vs M, both for the generator and for a successful attack
- the code for your implementations (Matlab/Octave, C, Java, or Python are all fine).

Laboratory exercise (1 student)

One-time password authentication

Consider the following (naïve) entity authentication protocol for A who wants to prove his identity to the verifier B.

A chooses a password p_{A} with up to $L_{\max}$ decimal digits and a one-way hash function $h : \{0,1,\ldots,9\}^{L_{\max}} \to \{0,1,\ldots,9\}^{L_{\max}}$. Let x_1 be the hash, and x_N be the N-fold hash of the (possibly zero-padded) password p_{A}, that is

$$x_1 = h([p_A, 0, \ldots, 0]) \quad , \quad x_N = \underbrace{h \circ h \circ \ldots h}_{N}([p_A, 0, \ldots, 0]) \; .$$

In the protocol setup phase, A securely delivers x_N to B. Then, in the n-th run of the protocol, $n = 1, \ldots, N-1$

1 $\mathsf{A} \to \mathsf{B}$: $u_1 = [\mathrm{id}_{\mathrm{A}}]$

2 B : $r_n \sim \mathcal{U}(\mathcal{R})$
$\mathsf{B} \to \mathsf{A}$: $u_2 = [n, r_n]$

3 A : $x_n = h(x_{n-1})$
A : $u_3 = [x_n, r_n]$
$\mathsf{A} \to \mathsf{B}$: u_3

4 B : checks whether $\underbrace{h \circ h \circ \ldots h}_{N-n}(x_n) = x_N$ and the received r_n is consistent with the transmitted one, and if so, B accepts A.

Your assignment is to:

1) implement the above protocol in your favourite language (Matlab/Octave, C, Java, or Python are all fine), and check its correctness;

2) identifiy its vulnerabilities and devise an attack that exploits them, under reasonable assumptions;

3) implement the attack and evaluate its success probability in dependence of the protocol parameters;

4) suggest improvements to the protocol and implement them.

Provide a description of your solution, with justification of your choices, the code for your implementations, and adequate discussion of the results.

General notes: Implementing your protocol may require you to use several cryptographic primitives, such as symmetric or asymmetric encryption/decryption, signing/verification, one-way functions, cryptographic hash functions, key generation, etc. Feel free to use any reasonable (and correct) implementation of such primitives you find for the language you've chosen, or even simplified models such as a (pseudo-)random oracle. The vulnerability of the protocols should not depend on a poor implementation of such primitives. Also, unlike for cryptographic keys, you should model the random choice of a password as (strongly) non uniform.

Laboratory exercise (2 students)

Physical Layer Secrecy example

Your aim is to implement a system that provides information theoretic secrecy and evaluate its security. The setup is that of a wiretap channel, with a legitimate transmitter (A), a legitimate receiver (B) and an eavesdropper (E), as follows:

- the channel from A to B is a memoryless binary symmetric channel (BSC) with error probability ε_{B}
- the channel from A to E is a memoryless binary symmetric channel (BSC) with error probability ε_{E}, and $\varepsilon_{\mathrm{E}} > \varepsilon_{\mathrm{B}}$
- errors in the two channels are independent

You will use a wiretap code with random binning, derived from a binary block (n, k) Hamming code. The message that you want to transmit reliably and secretly is made of a binary word u of length s, with $s \leq k$, randomly and uniformly generated. The 2^k codewords of n bits in a binary (n, k) Hamming code are to be partitioned into 2^s disjoint subsets, each with 2^{k-s} codewords, and each subset is associated to one of the possible secret words. Then, for transmission, the secret word u is mapped randomly and uniformly to a codeword x in its associated subset. After transmission, B will receive a corrupted version y, and E a different corrupted version z of the transmitted codeword.

1. Implement the (probabilistic) wiretap encoder as described above, the binary channel, and the corresponding (deterministic) decoder.
2. Fix the Hamming code with $n = 7, k = 4$, and the channels with $\varepsilon_{\mathrm{E}} = 1/5$, $\varepsilon_{\mathrm{B}} = 1/50$. For each possible value of $s \in \{1, 2, 3, 4\}$, evaluate through extensive simulation:
   (a) the *reliability* of the system, in terms of bit error probability on the secret word u as decoded by B
   (b) the *secrecy* of the system, in terms of bit error probability of the secret word u as decoded by E
   (c) the *secrecy* of the system, in terms of mutual information between the secret message u and the corrupted n-bit word z received by E

You will turn in

- a short report (a few pages) including:
  - a description of your implementations (encoder, decoder and channel), with adequate justification to your choices
  - a plot, or a table of results for the evaluation of system reliability and secrecy.
- the code for your implementations (Matlab/Octave, C, Java, or Python are all fine).

Laboratory exercise (1 student)

Physical Layer Secrecy over parallel Gaussian channels

Your aim is to implement the optimal power allocation scheme for Physical Layer Secrecy over a set of parallel Gaussian channels with known main and eavesdropper channel gains, and a bound on the maximum transmission power, as described in the slides *Secrecy rates for multiple channels*. Your algorithm should also compute the achievable secrecy rates with the given power bound, and the secrecy capacity in the infinite power limit. You are allowed to make use of any programming language of your choice (Matlab/Octave, C, Java, or Python are all fine)

Your algorithm must:

1. take as input the values of the SNR coefficients $\Lambda_{\mathrm{B},i}$ $\Lambda_{\mathrm{E},i}$ and of the maximum total power $P_{\max}$, as a list of numbers in floating point exponential representation (e.g., `1.25e-3` for $1/800 = 0.00125$) separated by spaces in the files `LambdaB.txt`, `LambdaE.txt` and `Pmax.txt` (only one value in this file), respectively
2. compute the amount of power P_i that should be allocated to each subchannel, and the corresponding achievable secrecy rates R_i
3. compute the total secrecy rate R_{s} and the high-SNR secrecy capacity C_{s}
4. output the values of allocated powers and achievable secrecy rates as a list of numbers in floating point exponential representation separated by spaces in the files `Powers.txt` and `Rates.txt`, respectively
5. output the values of the total secrecy rate and high-SNR secrecy capacity as numbers in floating point exponential representation in the files `TotalRate.txt`, `Capacity.txt`, respectively

You will turn in

- a short report (a few pages maximum) including:
  - a description of your implementation, with adequate justification to your choices
  - some example plots of significant cases showing: subchannel gains, power allocation values across the subchannels, achievable secrecy rate for each subchannel.
- the code for your implementation

Laboratory exercise (2 students)

Universal hashing with Toeplitz matrices

In discussing *universal hashing* we stated that binary Toeplitz matrices make a universal hashing class.

Formally, let $\text{Toep}(m, n, \boldsymbol{a})$ denote the $m \times n$ Toeplitz-like matrix $\boldsymbol{A}$ where the general entry A_{ij} is taken cyclically from the vector $\boldsymbol{a} = [a_0, \ldots, a_{p-1}]$ according to the rule

$$A_{ij} = a_{(m-i+j-1) \bmod p} \quad , \quad i = 1, \ldots, m \quad , \quad j = 1, \ldots, n$$

and write the set of all binary $\ell_t \times \ell_u$ Toeplitz matrices with ℓ_k elements as

$$\mathcal{F} = \left\{ \text{Toep}(\ell_t, \ell_u, \boldsymbol{k}),\, \boldsymbol{k} \in \mathbb{B}^{\ell_k} \right\} \ .$$

Then we stated that

> *If* $\ell_k \geq \ell_t + \ell_u - 1$, $\mathcal{F}$ *is a universal hashing class* $\mathbb{B}^{\ell_u} \to \mathbb{B}^{\ell_t}$.

Therefore, $\mathcal{F}$ can be used to provide unconditionally secure authentication and integrity protection for a message of ℓ_u bits, with a tag of ℓ_t bits, so that the success probability for any forging or illegitimate modification attack is upper bounded by $\varepsilon = 1/2^{\ell_t}$.

In this exercise *you are asked* to:

1. prove the above statement by numerical evaluation, for several values of ℓ_u and ℓ_t (you may wish to stick to small values for computational reasons)
2. find whether the above statement is true also for $\ell_k < \ell_t + \ell_u - 1$ that is, if shorter keys are used while keeping the same lengths for message and tag, by trying several values of ℓ_u, ℓ_t and ℓ_k
3. implement an authentication and integrity protection scheme for the transmission of a message of ℓ_u bits that uses the set $\mathcal{F}$ for general values of ℓ_t, ℓ_u, ℓ_k
4. build and simulate a forging attack against one of the systems in point 3, that achieves success with probability larger than $1/2^{\ell_t}$
5. build and simulate an illegitimate modification attack against one of the systems in point 3, that achieves success with probability larger than $1/2^{\ell_t}$

You will turn in, through the assignment Moodle page

- a short report (a few pages) including:
  - a description of your implementation of the hashing functions, the authentication and integrity protection scheme, and the attacks, with adequate justification to your choices;
  - the numerical results answering questions 1 and 2
  - numerical results or plots showing that the success probability of the attacks built in answering questions 4 and 5 are clearly larger than $1/2^{\ell_t}$
- the code for your implementation, numerical evaluation and/or simulations (Matlab/Octave, C, Java, or Python are all fine).

Laboratory exercise (1 student)

Zero knowledge authentication protocol

Consider the following (naïve) entity authentication protocol. A wants to authenticate himself to B, with whom he shares a secret key $k \in \{0,1\}^\ell$. B employs the following N-round protocol, which does not require A to send k over the channel.

For $n = 1$ to N

$\boxed{n.1}$ B : generates a challenge $x_n \sim \mathcal{U}(\{0,1\}^\ell)$

$\mathsf{B} \to \mathsf{A}$: x_n

$\boxed{n.2}$ A : calculates $y_n = x_n \oplus k$ and $b_n = \begin{cases} 1 & \text{if } y_n \text{ has an even number of ones} \\ 0 & \text{if } y_n \text{ has an odd number of ones} \end{cases}$

$\mathsf{A} \to \mathsf{B}$: b_n

$\boxed{n.3}$ B : checks b_n. If b_n is incorrect, the authentication is rejected

If b_n is correct for all $n = 1, \ldots, N$, the authentication is accepted

Your assignment is to:

1) implement the above protocol in your favourite language (Matlab/Octave, C, Java, or Python are all fine), and check its correctness;
2) identifiy its vulnerabilities and devise an attack that exploits them, under reasonable assumptions;
3) implement the attack and evaluate its success probability in dependence of the protocol parameters;
4) suggest improvements to the protocol and implement them.

Provide a description of your solution, with justification of your choices, the code for your implementations, and adequate discussion of the results.

General notes: Implementing your protocol may require you to use several cryptographic primitives, such as symmetric or asymmetric encryption/decryption, signing/verification, one-way functions, cryptographic hash functions, key generation, etc. Feel free to use any reasonable (and correct) implementation of such primitives you find for the language you've chosen, or even simplified models such as a (pseudo-)random oracle. The vulnerability of the protocols should not depend on a poor implementation of such primitives. Also, unlike for cryptographic keys, you should model the random choice of a password as (strongly) non uniform.