# Trabalhos Práticos EC 2021/2022

# Estruturas Criptográficas 2021/2022

## Datas de Entrega dos TP's

| **Trabalho** | **Data da demonstração** | **Comentários** |
|---|---|---|
| TP0 | 9 e 10 de Março | Em função do estado da pandemia as apresentações |
| TP1 | 30 e 31 de Março | realizam-se presencialmente ou via Zoom |
| TP2 | 4 e 5 de Maio | |
| TP3 | 1 e 2 de Junho | |
| **Recurso** | época de exames | |

## Funcionamento

1. Os alunos participam na avaliação prática integrados em grupos. A constituição dos vários grupos de trabalho é feita individualmente por cada aluno na folha de cálculo "Grupos e Slots".
2. Cada grupo contém 2 alunos ou excepcionalmente 3. Nos grupos com 3 alunos, a nota do trabalho é penalizada em $5\%$. Devido ao elevado número de inscrições não é possível existir participação individual não integrado num grupo.
3. Os trabalhos têm a forma de num "notebook" Jupyter distinto para cada um dos problemas indicados. Cada notebook deve **resumidamente**
   a. descrever o problema e a abordagem usada para o resolver
   b. apresentar o código Python que resolve o problema
   c. apresentar exemplos e testes de aplicação.
4. A entrega do trabalho tem a forma de uma discussão oral de 30 minutos com todos os elementos do grupo, e inclui a demonstração da boa execução do código.
5. Os "notebooks" executáveis e uma cópia PDF de cada um, devem ser previamente enviados via e-mail ao responsável da disciplina (@José Manuel V ).

6. A classificação é específica de cada elemento do grupo segundo a perceção que o avaliador tem da contribuição de cada um.
7. Demonstrações presenciais (30 min por grupo) nos dias indicados; entrega via email até às 24:00 da segunda-feira anterior. A inscrição no horário de entrega é feita na folha de cálculo "Grupos e Slots".

## Exame de Recurso

A avaliação é feita exclusivamente por trabalhos práticos. Dentro deste princípio um aluno pode realizar, em época de exame de recurso, um trabalho que substitua um dos trabalhos que deveriam ser entregues durante o semestre. A essa prova aplica-se os seguinte princípios

1. O trabalho é específico de cada aluno e comunicado pelo docente via e-mail na véspera do exame.
2. O trabalho deve ser completado até às 17 horas dia do exame de recurso e enviado por e-mail ao docente da disciplina.
3. Imediatamente a seguir ao envio ou no dia seguinte realiza-se uma apresentação do trabalho e sua discussão oral, nos moldes usados nos trabalhos do semestre.

## Ferramentas

1. **Python**
   a. Instalar Python versão 3.7 ou posterior
      i. Instalar Jupyter com suporte *ipykernel*
      ii. Instalar *scipy*, *matplotlib*, *numpy ;* opcionalmente instalar (recomendável para *big-data* e *machine-learning)* as packages *pandas* e *scikit-learn*; .
   b. Em alternativa a estas instalações individuais instalar Python e as restantes ferramentas via a distribuição Anaconda.
   c. Instalar a *package* **Cryptography** *(já incluído no Anaconda).*
   d. Instalar um IDE Python, por exemplo o PyCharm CE. (opcional)
2. **Sagemath**
   a. No MacOs e Linux's, com o Anaconda instalado, instalar o Sagemath via conda.
   b. Em alternativa instalar Sagemath a partir de https://www.sagemath.org/download.html. Consoante o sistema operativos existem várias opções. Nomeadamente:

i. No Mac OS é conveniente instalar a versão "app". Se instalar a versão linha de comando, ligá-la como um *kernel* do Jupyter.
ii. No Windows existe um comando ".exe" que funciona com o seu próprio Python.
iii. É também possível instalar (com um ficheiro OVA) na OracleVM VirtualBox incluindo o Extention Pack.

---

# Trabalho Prático 0

Use a package **Criptography** para

1. Criar um comunicação privada assíncrona entre um agente *Emitter* e um agente *Receiver* que cubra os seguintes aspectos:
   a. Autenticação do criptograma e dos metadados (*associated data*). Usar uma cifra simétrica num modo **HMAC** que seja seguro contra ataques aos "nounces" .
   b. Os "nounces" são gerados por um gerador pseudo aleatório (PRG) construído por um função de hash em modo XOF.
   c. O par de chaves $\text{cipher\_key}, \text{mac\_key}$ , para cifra e autenticação, é acordado entre agentes usando o protocolo **DH** com autenticação dos agentes usando assinaturas **DSA**.

2. Criar uma cifra com autenticação de meta-dados a partir de um PRG
   a. Criar um gerador pseudo-aleatório do tipo XOF ("extened output function") usando o SHAKE256, para gerar uma sequência de palavras de 64 bits.
      i. O gerador deve poder gerar até um limite de $2^n$ palavras ($n$ é um parâmetro) armazenados em *long integers* do Python.
      ii. A "seed" do gerador funciona como $\text{cipher\_key}$ e é gerado por um KDF a partir de uma "password" .
      iii. A autenticação do criptograma e dos dados associados é feita usando o próprio SHAKE256.
   b. Defina os algoritmos de cifrar e decifrar : para cifrar/decifrar uma mensagem com blocos de 64 bits, os "outputs" do gerador são usados como máscaras XOR dos blocos da mensagem.
   Essencialmente a cifra básica é uma implementação do "One Time Pad".

3. Compare experimentalmente a eficiência dos dois esquemas de cifra.

# Trabalho Prático 1

1. Use o "package" **Cryptography** para
   a. Implementar uma AEAD com "Tweakable Block Ciphers" conforme está descrito na última secção do texto +Capítulo 1: Primitivas Criptográficas Básicas. A cifra por blocos primitiva, usada para gerar a "tweakable block cipher", é o AES-256 ou o ChaCha20.
   b. Use esta construção para construir um canal privado de informação assíncrona com acordo de chaves feito com "X448 key exchange" e "Ed448 Signing&Verification" para autenticação dos agentes. Deve incluir uma fase de confirmação da chave acordada.

2. Use o **SageMath** para,
   a. Construir uma classe Python que implemente um **KEM- RSA**. A classe deve
      i. Inicializar cada instância recebendo o *parâmetro de segurança* (tamanho em bits do módulo **RSA**) e gere as chaves pública e privada.
      ii. Conter funções para encapsulamento e revelação da chave gerada.
   b. Construir, a partir deste KEM e usando a transformação de Fujisaki-Okamoto, um PKE que seja IND-CCA seguro.

3. Use o **Sagemath** para
   a. Construir uma classe Python que implemente o **EdCDSA** a partir do "standard" **FIPS186-5**
      i. A implementação deve conter funções para assinar digitalmente e verificar a assinatura.
      ii. A implementação da classe deve usar uma das "Twisted Edwards Curves" definidas no standard e escolhida na iniciação da classe: a curva "edwards25519" ou "edwards448".

Consultar também a diretoria EcDSA para informação adicional sobre o RFC 8032 que propõe o standard para o esquema EdDSA assim como os parâmetros das curvas "edwards25519" e "edwards448".

# Trabalho Prático 2

Este trabalho é dedicado às candidaturas finalistas ao concurso NIST Post-Quantum Cryptography na categoria de criptosistemas PKE-KEM.

De momento estão selecionadas 4 candidaturas finalistas (Classical McEliece, NTRU, KYBER e SABER) mas também estão selecionadas 5 outras candidaturas com oportunidade de virem a ser selecionadas. Deste último grupo destacamos BIKE por ser um criptosistemas que, tal como o Classical McEliece, é baseado em problemas de códigos mas é muito mais simples de implementar.

- O objetivo deste trabalho é a criação de 3 protótipos em **Sagemath** de três técnicas representativas cada uma delas das principais famílias de criptosistemas pós-quânticos: **BIKE** ("code based"), **NTRU** ("lattice based") e **KYBER** ("LWE based").
- Para cada uma destas técnicas pretende-se implementar um KEM, que seja IND-CPA seguro, e um PKE que seja IND-CCA seguro.
- A descrição, outra documentação e implementações em C/C++ das candidaturas aqui referidas pode ser obtida na página do concurso NIST ou na diretoria Dropbox da disciplina: Docs/NIST-PQC-ROUND3-PKE

# Trabalho Prático 3

1. Este problema pretende implementar algumas das candidaturas ao concurso NIST Post-Quantum Cryptography na categoria de esquemas de assinatura digital. Ver também a directoria com a documentação.

   O objetivo do problema é criar protótipos em Sagemath para os algoritmos Dilithium e Rainbow.

2. Pretende-se implementar em Sagemath o algoritmo de Schnorr (ver +Capítulo 8: Reticulados. Problemas "Standard" . Redução Linear. ) para factorizar inteiros a partir de uma solução aproximada do problema BDD/CVP em reticulados.