A classe **MasterBH**, que implementa a interface **Master**, é incumbida tanto de gerenciar o registro e a exclusão de escravos como também de controlar o fluxo de divisão dos ataques requisitados pelo cliente.

Ao ser invocado pelo cliente com a operação master.attack(ciphertext, knowntext) para processar um ataque de dicionário ao texto cifrado, o mestre deve subdividir a tarefa entre cada escravo. Após dividir igualmente espaçado o dicionário é executado slave.startSubAttack(cipherText, knownText, initialWordIndex, finalWordIndex, callbackInterface) de forma não bloqueante. Logo em seguida uma thread é invocada para vigiar cada um dos escravos, a espera de checkpoints que confirmem que o escravo não está ocioso. Após iniciar a thread o mestre entra em estado de espera até que seja acordado então ao receber a confirmação de que todos os escravos terminaram.

O mestre também é responsável por reconhecer quando um escravo realiza um checkpoint e também quando o escravo acha um **Guess**.

Ao receber um Guess o mestre deve adicionar o guess a lista de guesses a serem retornados para o cliente.

Ao receber um checkpoint o mestre atualiza as informações do escravo, mantendo assim o seu atual posicionamento no dicionário e também atualizando quando foi a última vez que o checkpoint ocorreu, por último é verificado se o dado checkpoint indica o final do ataque, caso indique o mestre é então acordado e a thread de vigia é destruida e a função de ataque retorna ao cliente.

A classe **SlaveAttack** é criada para garantir uma granularidade maior de encapsulamento ao processamento das atividades dos escravos. Ao ser iniciado um ataque, o mestre cria uma instância da classe e então a adiciona a uma lista global de ataques, mantendo assim referência a todos os ataques vigentes.

attack = new SlaveAttack(slave, initialWordIndex, finalWordIndex, this);

attacks.add(attack);

O objetivo dessa classe, além de manter o encapsulamento, é garantir consistente de informações de acesso, como a última vez em que o escravo deu o seu checkpoint e em que índice o escravo está.

A classe **SlaveBH** que implementa a interface **Slave**, define o escravo utilizado pelo mestre para realização do ataque particionado de dicionário. O escravo deve por definição procurar pelo nome *mestre* na tabela de registro no ***host*** informado em tempo de execução.

slave**.**slaveManager **=** (**SlaveManager**) registry**.**lookup("mestre");

Caso não encontrado o escravo deve tentar de novo a cada 30 segundos, também é necessário que o escravo pergunte de tempos em tempos se o mestre está disponível pelo registro, caso não esteja volte a procurar o mestre a cada 30 segundos.

slave**.**reRegistrationHook(host);

A principal atividade de um escravo é realizar a operação startSubAttack(ciphertext, knowntext, initialwordindex, finalwordindex,cb) de forma não bloqueante. Para realizar tal operação o escravo cria uma **Thread** que irá realiza o ataque e então à inicializa, retornando assim ao mestre o fim da função.

Ao mesmo tempo é também inicializada uma **Thread** que roda de tempos em tempos (10 segundos definido pela implementação vigente) uma função que envia um checkpoint ao mestre, informando em que índice aquele ataque se encontra.

checkpointScheduler**.**scheduleAtFixedRate(automaticCheckpoint, 10, 10, **SECONDS**);

Ao final do ataque outro checkpoint é enviado indicando assim o final.