

How Key recovery is performed using the PAT tool

```
public HashMap<Object, Object> CPA(int keysize) throws IOException {
    this.key = new int[keysize];
    LOGGER.info("Running Analysis...");
    this.initTraceMatrix();
    long start = System.currentTimeMillis();
    for (int i = 1; i <= keysize; ++i) {
        this.initHypothesis_MCU8_AES128(i);
        this.findCorrelation();
        this.key[i - 1] = this.findKey();
    }
    long end = System.currentTimeMillis();
    LOGGER.info("Analysis Complete.");
    double timetaken = (end - start) / 1000L;
    String strkey = "";
    String t = "";
    for (int i = 0; i < keysize; ++i) {
        strkey = strkey + ((t = Integer.toHexString(this.key[i]).toUpperCase()).length() < 3 ? "0" + t : t);
        System.out.println("t = " + t + " \t strkey = " + strkey);
    }
    System.out.println(strkey.trim());
    System.out.println("Total Time: " + timetaken + " seconds");
    HashMap<Object, Object> retVal = new HashMap<Object, Object>();
    retVal.put("key", strkey);
    retVal.put("time", timetaken);
    return retVal;
}
```

InitTraceMatrix();

Initializes a matrix based on Trace count and Number of Trace Points,
Obtains Plaintext and Ciphertext.

```
for (int i = 1; i <= keysize; ++i) {
    this.initHypothesis_MCU8_AES128(i);
    this.findCorrelation();
    this.key[i - 1] = this.findKey();
}
```

Iterate through all the keys based on number of keys (keysizes), run the following functions:

//Gather Hypothesis

initHypothesis_MCU8_AES128();

// Find Correlation between actual power trace curve and estimated curve

this.findCorrelation();

// Find Best guess Key and assign to array

this.key[i - 1] = this.findKey();

```

public void initHypothesis MCU8 AES128(int byteNumber) throws IOException {
    int i;
    int[] keyHyp = new int[256];
    for (i = 0; i < 255; ++i) {
        keyHyp[i] = i;
    }
    this.hypothesis = new int[this.plainText.length][keyHyp.length];
    for (i = 0; i < this.plainText.length; ++i) {
        String P = this.plainText[i].substring(2 * (byteNumber - 1), 2 * byteNumber);
        for (int j = 0; j < keyHyp.length; ++j) {
            this.hypothesis[i][j] = this.HW(this.SBOX(this.hex2int(P) ^ keyHyp[j]));
        }
    }
}
}

```

Generate Hypothesis by using the formulae:

$$\text{this.HW}(\text{this.SBOX}(\text{this.hex2int}(P) \wedge \text{keyHyp}[j]));$$

with this.HW returning Integer.bitCount(n);

with this.SBOX returning the SBOX' value;

based on the XOR-ed value of (hypothesized key which is between 0 to 255 and Plaintext)

by doing so, this.HW generates the estimated power trace level.

findCorrelation: Compares hypothesized trace level with actual trace level implemented with the help with Correlation function

```

public double Correlation(double[] xs, double[] ys) {
    double sx = 0.0;
    double sy = 0.0;
    double sxx = 0.0;
    double syy = 0.0;
    double sxy = 0.0;
    int n = xs.length;
    for (int i = 0; i < n; ++i) {
        double x = xs[i];
        double y = ys[i];
        sx += x;
        sy += y;
        sxx += x * x;
        syy += y * y;
        sxy += x * y;
    }
    double cov = sxy / (double)n - sx * sy / (double)n / (double)n;
    double sigmax = Math.sqrt(sxx / (double)n - sx * sx / (double)n / (double)n);
    double sigmay = Math.sqrt(syy / (double)n - sy * sy / (double)n / (double)n);
    return cov / sigmax / sigmay;
}

```

findKey: Basically compares the data point with the highest correlation