Unixcrypt Multi-block Analysis

Charlie Gerrie

Dalhousie University

Abstract

This research introduces a method for multi-block analysis of the unix crypt cipher. We extend a single-block algorithm previously developed for this cipher. Taking advantage of how block keys are related to each other, we collect information about the key used to generate them. We then use statistical methods to pick the most likely key. Thus with sufficiently many blocks we can recover the original key and decrypt the ciphertext completely.

1 Preliminaries

- We consider the permutations of $\mathbb{Z}_{256}$ to be all functions $\mathbb{Z}_{256} \to \mathbb{Z}_{256}$. We make use of their group structure under composition, as well as their group action on $\mathbb{Z}_{256}$ by application. We use multiplicative notation for group operations, and exponentiation will represent repeated multiplication (e.g. $\sigma^3 = \sigma\sigma\sigma$).
- We call a pair of elements (a, b) in $\mathbb{Z}_{256}$ a *wire*, with a being the *start-point* and b being the *end-point*. A wire (a, b) is said to be *in* a permutation σ if $\sigma(a) = b$. Thus a wire represents a single image-preimage pair of a permutation.
- We associate the elements of $\mathbb{Z}_{256}$ with bytes when considering the implementation of the cipher and breaking algorithm. We also associate them with single characters, which in practice is done using ascii or a similar character encoding.
- Multisets will be used in a manner which intuitively expresses a set which can have multiple copies of the same element. Formally, a multiset X of cardinality $|X|$ is the equivalence class of $|X|$-tuples up to permutations. Thus sorting the elements yields a unique representation. We can assign the elements of this sorted tuple indices and treat them like elements of a sequence. An example of this could be letting x_i be the ith element of the sorted tuple. We say an element is *in* a multiset if it appears at least once in the tuple. The *multiplicity* of an element is the number of times it appears in the list.
- Sequences will be indexed starting at 0.
- We will make use of a commutative diagram [1]. Such a diagram is composed of arrows, representing permutations, meeting at vertices. A path of several arrows represents the composition of the permutations of its constituent arrows. Any two paths with the same start-point and end-point are equivalent.

2 The Cipher

The unixcrypt cipher key is defined by two permutations of $\mathbb{Z}_{256}$ denoted by the characters τ and σ. τ must be a 256-cycle. σ must be self-inverse ($\sigma^2 = \mathtt{id}$) and have no fixed points ($\sigma(x) \neq x$), or in other words it must be entirely composed of 2-cycles.

When using the cipher we will consider two bodies of text, the plaintext and the ciphertext. Both represent sequences of elements in $\mathbb{Z}_{256}$. The ciphertext will be produced from the plaintext by the cipher defined in equation (1). We partition these sequences into segments of 256 elements known as *blocks*. Thus consider p_i, the ith character of the plaintext. Let $i = 256j + k$, where $0 \leq k < 256$. Then p_i is part of the jth block of the plaintext, and is the kth character within that block.

We can then encode p_i into the ith character of the ciphertext using this equation:

$$c_i = \kappa^{-k}\tau^{-j}\sigma\tau^{j}\kappa^{k}p_i \tag{1}$$

where κ is the successor function $\kappa(x) = x + 1 \mod 256$.

Note that this cipher is symmetric-key, meaning it uses the same key for encryption and decryption. Also, the encryption and decryption processes are the exact same. This follows from equation (1) and the fact that $\sigma^2 = \texttt{id}$.

Notice that the exponent on τ is only incremented when j is incremented. This happens every 256 characters. Thus we define the jth *block key* σ_j by:

$$\sigma_j = \tau^{-j}\sigma\tau^j \tag{2}$$

This combines both the σ and τ parts of the key into one, giving us a single permutation for encoding the jth block. This will be important for using the information from the single-block analysis. Note that $\sigma_0 = \sigma$.

3 Single-Block Analysis

Previous work on this cipher has produced a method of single-block analysis [2]. This looks at each block individually and tries to find, or in practice approximates, the block keys for each block. Thus, we obtain approximations of each block key σ_j, which we will denote $\hat{\sigma}_j$.

Each block only consists of 256 characters, so on average every wire is used just once. In practice what this means is that the single-block analysis can successfully reproduce about half of the wires in each block key. It also makes a small number of incorrect guesses, and leaves the rest of the wires unguessed. This last part is insignificant when the single-block analysis is performed by itself, but to perform the multi-block analysis we will need the block key approximations to be complete permutations so we can perform calculations with them. Thus, to "fill out" the permutations we assign the unguessed wires at random. This can be done efficiently with a Fisher-Yates shuffle.

4 Multi-block Analysis

The input to the algorithm is a collection of block key approximation. We get these by running the single-block analysis on the ciphertext first.

output

goal

5 Coincidences

We define an indicator function δ taking two arguments x and y

$$\delta(x, y) = \begin{cases} 1 & \texttt{if } x = y \\ 0 & \texttt{if } x \neq y \end{cases}$$

Then for any two multisets X and Y with elements x_i and y_j, where $0 \leq i < |X|$ and $0 \leq j < |Y|$, we define their *coincidence* as:

$$\texttt{coinc}(X;Y) = \sum_{i,j} \delta(x_i, y_j) \tag{3}$$

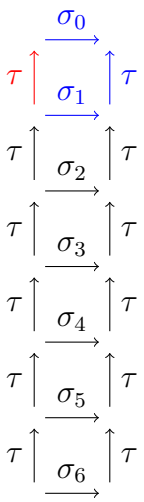
with the sum being over all possible pairs of i and j.

To demonstrate the principle of what we will be doing, let us consider this example. You have two bags of size k and k^2 each filled with labelled objects. If you didn't know which bag you had, could you *guess* only by sampling from the bags and counting the number of coincidences in the objects drawn? The answer is yes. If we pulled an object out the the bag, recorded which one it was, returned it, then drew another item, what would be the probability they were the same one? Well if we had the bag with k objects it would be $\frac{1}{k}$, but if we had the bag with k^2 objects it would be $\frac{1}{k^2}$. These values will be very different, so by sampling many times we can see whether the probability that we get two matching objects is closer to $\frac{1}{k}$ or $\frac{1}{k^2}$ and thus determine which bag we probably have.

5.1 Coincidences and the Algorithm

Figure 1 shows a commutative diagram derived from equation (2). Of particular note is the equivalence of τ and $\sigma_1\tau\sigma_0^{-1}$, marked in red and blue respectively. Thus, if we conjecture that (a, b) is a wire in τ then it must also be a wire in $\sigma_1\tau\sigma_0^{-1}$. But the wire in $\sigma_1\tau\sigma_0^{-1}$ is really three wires: a wire $(a, \sigma_1(a))$ in σ_1, a wire $(\sigma_1(a), \sigma_0(b))$ in τ, and a wire $(\sigma_0(b), b)$ in σ_0^{-1}. Thus, if (a, b) is a wire in τ then $(\sigma_1(a), \sigma_0(b))$ must also be a wire in τ. In general, all wires $(\sigma_{i+1}(a), \sigma_i(b))$ must be in τ. Thus, the *wire consequences* of (a, b) are all the wires that must also be in τ if (a, b) is.

Figure 1



Definition 1 (Wire Consequences)**.** For a wire (a, b), we have

$$\texttt{conseq}(a, b) = \{\!\{\hat{\sigma}_i(a), \hat{\sigma}_{i+1}(b)) : 0 \leq i < N - 1\}\!\} \tag{4}$$

where N is the number of $\hat{\sigma}$'s. This is a multiset.

TODO Explain why coincidences

Definition 2 (Wire-to-wire Coincidences)**.** If we consider two wires (a, b) and (c, d), then we have

$$\texttt{coinc}(a, b, c, d) = \sum_{i=0}^{255} \sum_{j=0}^{255} \texttt{conseq}(a, b)_{i,j} \texttt{conseq}(c, d)_{i,j} \tag{5}$$

where $\texttt{conseq}(a, b)_{i,j}$ is the multiplicity of the wire (i, j) in the consequences of (a, b). This is a count of all the possible pairs of similar elements in the two wires consequence multisets.

Figure 2

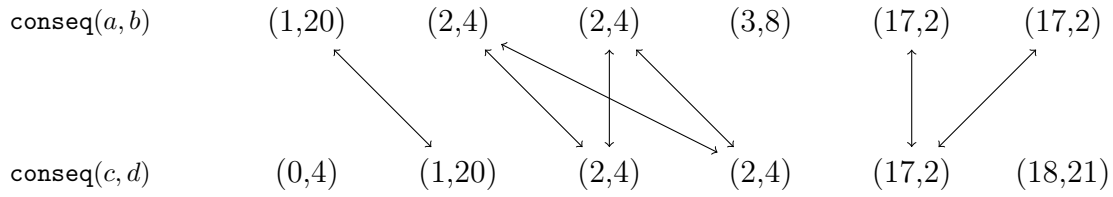


Figure 2 shows how we count the possible pairs. This example would be expressed as an expansion of equation (5) as follows:

$$\begin{aligned}
\texttt{coinc}(a, b, c, d) =& \sum_{i=0}^{255} \sum_{j=0}^{255} \texttt{conseq}(a, b)_{i,j} \texttt{conseq}(c, d)_{i,j} \\
=& \texttt{conseq}(a, b)_{1,20} \texttt{conseq}(c, d)_{1,20} + \texttt{conseq}(a, b)_{2,4} \texttt{conseq}(c, d)_{2,4} \\
&+ \texttt{conseq}(a, b)_{17,2} \texttt{conseq}(c, d)_{17,2} \\
=& (1)(1) + (2)(2) + (2)(1) \\
=& 7
\end{aligned} \tag{6}$$

with all other terms in the sums evaluating to zero.

Finally, it should be noted that wire-to-wire coincidences are always non-negative integers.

Definition 3 (Coincidences Distribution)**.** Given a wire, we can define a discrete distribution of the value of the wire-to-wire coincidences between that wire and another randomly chosen wire.

Here is an example the coincidences distribution for a wire (75,29), encoding the probability of each number of wire-to-wire coincidences between (75,29) and a randomly chosen wire (c,d).

X	0	1	2	...
$P(\texttt{coinc}(75, 29, c, d) = X)$	0.933	0.0648	0.00224	...

Notice that it is very likely that the number of wire-to-wire coincidences is 0, and that the probability falls of sharply as the number of coincidences increases. Note that in practice we generate this distribution by sampling the wire-to-wire coincidences between the given wire (a, b) and wires (c, d) that satisfy the following conditions: $a \neq b$ and $c \neq d$ because τ is a 256-cycle and cannot have fixed points, and $a \neq c$ and $b \neq d$ because then they would be mutually exclusive. There are 64771 such wires.

Now to reconstruct τ, we need to find all 256 of its wires. To do this, we calculate the coincidences distribution for all possible wires, of which there are 256×255 since wires in τ cannot connect to themselves. Then for all 256 wire start-points, we will pick the best end-point using the coincidences distribution. This will give us our guess at τ.

6 Likelihood

6.1 Likelihood and the Algorithm

To rigorously determine which distribution our sample distribution fits, we will be using a concept from statistics and probability theory known as *likelihood*. If we consider two events A and B, then likelihood is related to conditional probability by the relation

$$\mathcal{L}(A|B) = P(B|A)$$

Now we have two potential distributions and we would like to know which one our sample distribution follows. Let D_1 and D_2 be the events that the sample distribution follows the two distributions and S is the event of getting the samples that we did. We would like to calculate $P(D_1|S)$ and $P(D_2|S)$ and then choose the distribution with the higher probability. Now it is easy to calculate $P(S|D_1)$ and $P(S|D_2)$. Assuming the samples are independent, they are simply the products of the pdf values for all the samples. Normally we would then use Bayes' theorem:

$$P(D_1|S) = \frac{P(S|D_1)P(D_1)}{P(S)}$$

But this would require us to know an *a priori* probability of the distribution and the samples. This we cannot know, so Bayes' theorem fails us.

This is where likelihood comes to the rescue. The probabilities we can calculate, $P(S|D_1)$ and $P(S|D_2)$, are precisely the likelihoods $\mathcal{L}(D_1|S)$ and $\mathcal{L}(D_2|S)$. Since we cannot choose the most *probable* distribution of the two, we choose the most *likely*. This is known as the maximum likelihood method [3].

Now our problem is that for each wire start-point we have 255 possible end-points. Each end-point defines a potential wire with a coincidences distribution. For each wire we consider the question "is this wire actually in τ or not?" Let the null hypothesis be that it is an incorrect guess and is not in τ, and the alternative hypothesis be that it is in τ. The next

step would be to construct hypothetical distributions for both cases, but constructing the distribution for the alternative hypothesis is complicated and there is a simpler way. We can easily construct a distribution for the null hypothesis, which we will call the *null distribution*, and we can choose the wire which is least likely to fit this distribution and fulfill the null hypothesis.

If a wire is not in τ, then the logic behind its wire consequences also being in τ does not apply. Its consequences are then essentially random, so they are chosen from all 256×256 pairs

TODO FINISH

Thus the null distribution depends on the number of blocks being analyzed since that determines the number of wire consequences each wire has.

X	0	1	2	...
$P(\texttt{coinc}(a,b,c,d) = X)$	0.965	0.0345	0.000618	...

TODO number of blocks required

7 Finding σ

We can restate equation (2) as follows

$$\sigma = \tau^j \sigma_j \tau^{-j}$$

Then once we have determined τ we can recombine it with the block key approximations from the single-block analysis to get a collection of approximations for σ. Then for each wire start-point we pick the end-point that occurs the most times in these approximations.

8 Conclusion

9 Appendix: Tail Density?

10 Appendix: Code?

APPENDIX FOR K sigmastotau DATA STRUCTURE IMPROVEMENT?

11 References

[1] http://mathworld.wolfram.com/CommutativeDiagram.html

[2] https://www.mathstat.dal.ca/∼selinger/unixcrypt-breaker/

[3] http://mathworld.wolfram.com/MaximumLikelihood.html