Cryptanalysis of splay tree based encryption

Jean-Philippe Aumasson

Nagravision SA, Switzerland

Abstract. We present a chosen-plaintext attack on KIST, a recently proposed encryption scheme based on splay trees. Our attack recovers a 128-bit key with approximately 2^{28} bit operations and fewer than 2^{19} chosen-plaintext queries.

Splay trees are a type of binary search search trees discovered by Sleator and Tarjan in 1983 [1, 2]. Splay trees are self-adjusting, in the sense that a tree modifies itself at each access to a node—the modification being called "splaying". In 1988, Jones proposed [3] a simplification of splay trees to compress data, and proposed to use splay trees to encrypt data using a similar method with as key a secret initial tree. The (rather old) word processor Lotus Ami Pro included a basic, deliberately insecure, version of splay tree based encryption to allow export [4]. As the basic version of splay tree based encryption is insecure, Jones proposed [5] two techniques to strengthen that scheme.

Recently, Wei and Zeng argued [6] that Jones' strengthened schemes are also insecure, and proposed a new encryption scheme based on splay trees, called KIST. Below we describe a chosen-plaintext attack for that new scheme. We refer to [6] for a description of KIST.

Our attack is based on the following observations:

- By doing 256 chosen-plaintext encryption queries with the bytes $0, 1, \ldots, 255$, one can determine the initial shape of the tree (that is, after initialization) and the value of its leaves, since the encoding of each byte gives the position of that byte (as a leaf) within the tree.
- Generalizing the above technique, one can determine the shape of the tree after encrypting any sequence of bytes, in 2^8 chosen-plaintext queries. Note that one does not recover the labels of the inner nodes, but only those of the leaves.
- Given the shape of the tree before and after a key injection step, one can determine the two inner nodes swapped, i.e., the j such that $x_j = K_{i+16} = K_i \oplus x_k$. One can also determine k, as it is the parent of the (known) last plaintext byte processed. If the parameter N is not maximal, then one can detect when no nodes were swapped (since key injection must change the position of at least two leaves).
- Each subkey word K_i, $i \geq 17$, has its jth bit, $1 \leq j \leq 8$, depending only on the jth bit of $K_{((i-1) \bmod 16)+1}$ and on the jth bits of at most $\lfloor (i-1)/16 \rfloor$ inner node labels. Moreover, the dependencies are fully XOR-linear.
- A (semi-rotation) splaying does not depend on the value of the inner nodes, but only on the shape of the tree. Given an initial tree shape and an encrypted byte, one can thus determine the new shape of the tree by making a sequence of semi-rotations.

The proposed attack goes as follows:

1. Determine the initial shape of the tree, by doing 256 chosen-plaintext queries.
2. Assign arbitrary labels $x_1, \ldots, x_{255}$ to each of the inner nodes.
3. Set $m \leftarrow \lambda$ (i.e., the empty message).
4. Initialize the equations counter $c \leftarrow 1$.
5. Initialize the loops counter $i \leftarrow 1$.
6. While $c < 255 + 16$ do
   (a) Select an arbitrary byte s_i.
   (b) Set $m \leftarrow m \| s_i$.
   (c) Determine the shape of the tree after encrypting m, by doing 256 chosen-plaintext queries of the form $m \| b$, where $b = 0, 1, \ldots, 255$.
   (d) Compare the shape obtained with the shape of the previous tree after splaying around s_i. If the trees are distinct (that is, key injection modified the tree), then
      i. Determine j and k such that the node $x_j = K_i \oplus x_k$ is used for key injection, and add this equation to the system of equations.
      ii. Set $c \leftarrow 1 + c$.
   (e) Set $i \leftarrow 1 + i$.
7. Simplify the system of equations by replacing K_{i+16}, $i \geq 1$ by an XOR between $K_{((i-1) \bmod 16)+1}$ and labels of inner nodes, as defined by the key generation and by the equations collected.
8. For each bit slice, solve the linear system of equations with as unknown the bits of $K_1, \ldots, K_{16}$ and of the inner nodes' labels, at a given position.

The complexity of the attack depends on the parameter N, which determines the frequency of a key injection and thus the number of iterations of the "while" loop. If N is maximal a new equation is obtained at each loop (and so $c = i$). One thus has to determine 272 tree shapes, where each one costs at most 2^8 chosen-plaintext queries. About 2^{16} chosen-plaintext queries are thus necessary to collect enough equations when N is maximal.

The analysis in [6] suggests that N should be such that at least 20 % of the key injections are effective. In this case, one will need approximately $1+5\times(255+16) = 1356$ tree shapes. One thus needs about 2^{18} chosen-plaintext queries.

In both cases, solving the linear systems costs about $8 \times 271^3 \approx 2^{28}$ bit operations. Slightly more than $255 + 16$ equations may be necessary, to deal with linearly dependent equations. For the 7-bit version proposed in [6, §4], solving the linear systems costs about $7 \times (127 + 16)^3 \approx 2^{25}$ bit operations.

References

1. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary trees. In: STOC, ACM (1983) 235–245
2. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**(3) (1985) 652–686
3. Jones, D.W.: Application of splay trees to data compression. Commun. ACM **31**(8) (1988) 996–1007
4. Jones, D.W.: Data compression and encryption algorithms. Accessed 9 November 2010 `www.cs.uiowa.edu/~jones/compress/`.
5. Jones, D.W.: Patching splay encryption to weaken chosen plaintext attacks. Accessed 9 November 2010 `www.cs.uiowa.edu/~jones/compress/plaintext.html`.
6. Wei, R., Zeng, Z.: KIST: A new encryption algorithm based on splay. Cryptology ePrint Archive, Report 2010/425 (2010) `http://eprint.iacr.org/`.