YAARX: Yet Another ARX Toolkit

0.1

Generated by Doxygen 1.7.6.1

Fri Mar 15 2013 01:24:33

Contents

1 A Toolkit for Analysis of ARX Cryptographic Algorithms

1.1 What is YAARX

YAARX is a set of programs for the differential analysis of ARX cryptographic algorithms. The latter represent a broad class of symmetric-key algorithms designed by combining a small set of simple operations such as modular addition, bit rotation, bit shift and XOR. More notable representatives of the ARX class of algorithms are the block ciphers FEAL, RC5, TEA and XTEA, the stream cipher Salsal20, the hash functions MD4, MD5, Skein and BLAKE as well as the recently proposed hash function for short messages SipHash.

1.2 How Does it Compare to Other ARX Tools

YAARX complements existing toolkits such as `ARXtools` and extends others, such as `The S-function Toolkit`. More specifically, YAARX is the first tool that provides means to search for differential trails in ARX algorithms in a fully automatic way. The latter has been a notoriously difficult task for ciphers that do not have S-boxes, such as ARX. Additionally, YAARX provides methods for the computation of the differential probabilities of various ARX operations (XOR, modular addition, bit shift, bit rotation) as well as of several larger components built from them.

1.3 What Can YAARX Do for Me

YAARX can help the cryptanalyst in the process of analyzing ARX-based constructions in at least two ways.

1.3.1 Using the Tools Directly

One way is to use the tools directly to compute differential probabilities for a target cipher. To this end YAARX provides a set of programs for the computation of the differential probabilities (DP) of several operations with user provided inputs. Such are for example the programs for computing the DP of modular addition, XOR, bit shift, bit rotation, etc.

A conceivable scenario in this category would be the case in which the cryptanalyst constructs a differential characteristic by hand and wants to estimate its probability by computing the probabilities of its corresponding differentials through the ARX operations. In this case YAARX can provide answer to questions such as:

- Given input differences da and db to an operation F, and an output difference dc, what is the probability of the differential $(da,\ db \rightarrow dc)$?
- Given input differences da and db to F, what is the output difference dc that has maximum probability?
- Given an input difference da and an input value b to F and an output difference dc, what is the probability of the differential $(da,\ b \rightarrow dc)$?
- Given input difference da and a set of input differences $\{db_0,\ db_2,\ db_3,\ dc_4\}$ to F, and an output difference dc, what is the probability of the differential $(da,\ \{db_0,\ db_2,\ db_3,\ db_4\} \rightarrow dc)$?
- etc...

The differences da, db and dc can be XOR or additive (ADD) differences and the operation F can either be one of the basic ARX operation, such as XOR, addition, etc. or a larger component e.g. a sequence of bit shift and XOR or of addition, rotation and XOR.

1.3.2 Modifying the Source Code

The second way in which YAARX can be useful would require a bit more effort and some programming literacy on the part of the cryptanalyst. The idea is, instead of directly using one of the YAARX tools, to first modify it according to ones' specific needs. This scenario is realistic in a case in which none of the YAARX tools is capable of directly solving a problem for a given target cipher.

Such a case is likely to occur for example when one wants to automatically search for differential trails in a given cipher. While YAARX supports a general strategy for automatic search of trails, that is potentially applicable to many ARX algorithms, it is implemented for two specific ciphers, namely TEA and XTEA. Since the algorithmic technique underlying this implementation is general, the latter can be applied to other ARX algorithms after respective modifications.

1.4 The YAARX Tools

A list of the tools provided by YAARX, together with their computaional complexities is given in the table below. For more details on a specific algorithm refer to the `documentation`.

Δ **Operator**	**Algorithm**	**Description**, (DP = differential probability, ADD = modular addition)	**Complexity**, (n = word size, bits)
$+$	$\mathrm{adp}^{\lll}$	The ADD DP of left shift (LSH).	$O(1)$
$+$	$\mathrm{adp}^{\ggg}$	The ADD DP of right shift (RSH).	$O(1)$
$\oplus$	$\mathrm{xdp}^{+}(da, db \rightarrow dc)$	The XOR DP (XDP) of ADD.	$O(n)$
$+$	$\mathrm{adp}^{\oplus}(da, db \rightarrow dc)$	The ADD DP (ADP) of XOR.	$O(n)$
$+$	$\mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow db)$	The ADD DP of XOR with one fixed input.	$O(n)$
$+$	$\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$	The ADD DP of XOR with three inputs.	$O(n)$
$+$	$\mathrm{adp}^{\ggg\oplus}$	The ADD DP of RSH followed by XOR.	$O(n)$
$+$	$\mathrm{eadp}^{F}(da \rightarrow dd)$.	The expected additive DP (EADP) of the F-function of TEA, averaged over all round keys and constants: $F(k_0, k_1, \delta \vert\, x) = ((x \lll 4) + k_0) \oplus (x + \delta) \oplus ((x \ggg 5) + k_1)$.	$O(n)$
$+$	$\mathrm{adp}^{F'}(k_0, k_1, \delta \vert\, da \rightarrow db)$	The additive DP (ADP) of a modified version of F': the F-function of TEA with the shift operations removed: $y = F'(k_0, k_1, \delta \vert\, x) = (x + k_0) \oplus (x + \delta) \oplus (x + k_1)$.	$O(n)$
$\oplus$	$\max_{dc}\ \mathrm{xdp}^{+}(da, db \rightarrow dc)$	The maximum XOR DP of ADD.	$O(n) \leq c \ll O(2^n)$
$+$	$\max_{dc}\ \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$	The maximum ADD DP of XOR.	$O(n) \leq c \ll O(2^n)$
$+$	$\max_{dc}\ \mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow dc)$.	The maximum ADD DP of XOR with one fixed input:	$O(n) \leq c \ll O(2^n)$
$+$	$\max_{dd}\ \mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$	The maximum ADD DP of XOR with three inputs:	$O(n) \leq c \ll O(2^n)$
$+$	$\max_{dd}\ \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$	The maximum ADD DP of XOR with three inputs, where one of the	$O(n) \leq c \ll O(2^n)$

1.5 Copyright

2 Directory Hierarchy

2.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

5 Directory Documentation

5.1 include/ Directory Reference

Files

- file adp-rsh-xor.hh

  Header file for adp-rsh-xor.cc: The ADD differential probability of right shift followed by XOR: $\mathrm{adp}^{\gg\oplus}$. .
- file adp-shift.hh

  Header file for adp-shift.cc: The ADD differential probability of left shift (LSH): $\mathrm{adp}^{\ll}$ *and right shift (RSH):* $\mathrm{adp}^{\gg}$. .
- file adp-tea-f-fk-ddt.hh

  Header file for adp-tea-f-fk-ddt.cc: Computing the full difference distribution table (DD-T) for the F-function of block cipher TEA by exaustive search over all inputs. Complexity $O(2^{2n})$. .
- file adp-tea-f-fk-noshift.hh

  Header file for adp-tea-f-fk-noshift.cc: The additive differential probability (ADP) of a modified version of the F-function of TEA with the shift operations removed. - Complexity $O(n)$. .
- file adp-tea-f-fk.hh

  Header file for adp-tea-f-fk.cc: The ADD differential probability of the F-function of TEA for a fixed key and round constants $\mathrm{adp}^F(k_0,k_1,\delta|\, da \rightarrow dd)$*, where* $F(k_0,k_1,\delta|\, x) = ((x \ll 4) + k_0) \oplus (x+\delta) \oplus ((x \gg 5) + k_1)$ *Complexity:* $O(n) < c \leq O(2^n)$. .
- file adp-xor-fi.hh

  Header file for adp-xor-fi.cc: The ADD differential probability of XOR with one fixed input (FI): $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow db)$. .
- file adp-xor-pddt.hh

  Header file for adp-xor-pddt.cc. Compute a partial difference distribution table (pDDT) for $\mathrm{adp}^{\oplus}$. .
- file adp-xor.hh

  Header file for adp-xor.cc: The ADD differential probability of XOR $\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.
  .
- file adp-xor3.hh

  Header file for adp-xor3.cc: The ADD differential probability of XOR with three inputs ($3\oplus$*):* $\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$. .
- file adp-xtea-f-fk.hh

  Header file for adp-xtea-f-fk.cc: The ADD differential probability of the F-function of XTEA for a fixed key and round constants $\mathrm{adp}^F(k,\delta|\, da \rightarrow dd)$*. Complexity:* $O(n) < c \leq O(2^n)$. .
- file common.hh

  Header file for common.cc. Common functions used accross all YAARX programs. .
- file eadp-tea-f.hh

  Header file for eadp-tea-f.cc. The expected additive differential probability (EADP) of the F-function of TEA, averaged over all round keys and constants: $\mathrm{eadp}^F(da \rightarrow dd)$*. Complexity:* $O(n)$. .
- file max-adp-xor-fi.hh

Header file for max-adp-xor-fi.cc. The maximum ADD differential probability of XOR with one fixed input: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow dc)$. .

- file max-adp-xor.hh

  Header file for max-adp-xor.cc. The maximum ADD differential probability of XOR: $\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$. .

- file max-adp-xor3-set.hh

  Header file for max-adp-xor3-set.cc. The maximum ADD differential probability of - XOR with three inputs, where one of the inputs satisfies a set of ADD differences: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$. .

- file max-adp-xor3.hh

  Header file for max-adp-xor3.cc. The maximum ADD differential probability of XOR with three inputs: $\max_{dd} \mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$. .

- file max-xdp-add.hh

  Header file for max-xdp-add.cc. The maximum XOR differential probability of ADD: $\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$. .

- file tea-add-ddt-search.hh

  Declarations for tea-add-ddt-search.cc. Automatic search for ADD differential trails in TEA using full DDT-s. .

- file tea-add-threshold-search.hh

  Header for tea-add-threshold-search.cc. Automatic search for ADD differential trails in block cipher TEA. .

- file tea-f-add-pddt.hh

  Header file for tea-f-add-pddt.cc. Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA. .

- file tea.hh

  Header file for tea.cc. Common functions used in the analysis of TEA. .

- file xdp-add-pddt.hh

  Header file for xdp-add-pddt.cc. Compute a partial difference distribution table (pDDT) for xdp^{+}. .

- file xdp-add.hh

  Header file for xdp-add.cc: The XOR differential probability of ADD $\mathrm{xdp}^{+}(da, db \rightarrow db)$. .

- file xdp-tea-f-fk.hh

  Header file for xdp-tea-f-fk.cc. The XOR differential probability (XDP) of the F-function of TEA for a fixed key and round constants: $\mathrm{xdp}^{F}(k_0, k_1, \delta | \, da \rightarrow dd)$. .

- file xdp-xtea-f-fk.hh

  Header file for xdp-xtea-f-fk.cc. The XOR differential probability (XDP) of the F-function of XTEA for a fixed key and round constants: $\mathrm{xdp}^{F}(k, \delta | \, da \rightarrow dd)$. .

- file xtea-add-threshold-search.hh

  Header file for xtea-add-threshold-search.cc. Automatic search for ADD differential trails in block cipher XTEA. .

- file xtea-f-add-pddt.hh

  Declarations for xtea-f-add-pddt.cc. Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher XTEA. .

- file xtea-f-xor-pddt.hh

  Header for xtea-f-xor-pddt.cc. Computing an XOR partial difference distribution table (pDDT) for the F-function of block cipher XTEA. .

- file xtea-xor-threshold-search.hh

  Header file for xtea-xor-threshold-search.cc. Automatic search for XOR differential trails in block cipher XTEA. .
- file xtea.hh

  Header file for xtea.cc. Common functions used in the analysis of block cipher XTEA. .

5.2 src/ Directory Reference

Files

- file adp-lsh-program.cc

  The probability $\mathrm{adp}^{\ll}$ *with user-provided input.*
- file adp-rsh-program.cc

  The probability $\mathrm{adp}^{\gg}$ *with user-provided input.*
- file adp-rsh-xor.cc

  The ADD differential probability of right shift followed by XOR: $\mathrm{adp}^{\gg\oplus}$.
- file adp-shift.cc

  The ADD differential probability of left shift (LSH): $\mathrm{adp}^{\ll}$ *and right shift (RSH):* $\mathrm{adp}^{\gg}$.
- file adp-tea-f-fk-ddt.cc

  Computing the full difference distribution table (DDT) for the F-function of block cipher TEA by exaustive search over all inputs. Complexity $O(2^{2n})$.
- file adp-tea-f-fk-noshift.cc

  The additive differential probability (ADP) of a modified version of the F-function of TEA with the shift operations removed. Complexity $O(n)$.
- file adp-tea-f-fk.cc

  The ADD differential probability of the F-function of TEA for a fixed key and round constants $\mathrm{adp}^{F}(k_0,k_1,\delta|\ da \rightarrow dd)$, *where* $F(k_0,k_1,\delta|\ x) = ((x \ll 4)+k_0) \oplus (x+\delta) \oplus ((x \gg 5)+k_1)$ *Complexity:* $O(n) < c \leq O(2^n)$.
- file adp-xor-fi-program.cc

  The probability $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ *with user-provided input.*
- file adp-xor-fi.cc

  The ADD differential probability of XOR with one fixed input (FI): $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(a,db \rightarrow db)$.
- file adp-xor-pddt.cc

  Compute a partial difference distribution table (pDDT) for $\mathrm{adp}^{\oplus}$.
- file adp-xor-program.cc

  The probability $\mathrm{adp}^{\oplus}$ *with user-provided input.*
- file adp-xor.cc

  The ADD differential probability of XOR $\mathrm{adp}^{\oplus}(da,db \rightarrow db)$.
- file adp-xor3-program.cc

  The probability ($\mathrm{adp}^{3\oplus}$ *) with user-provided input.*
- file adp-xor3.cc

  The ADD differential probability of XOR with three inputs ($3\oplus$*):* $\mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$.

- file adp-xtea-f-fk.cc

  The ADD differential probability of the F-function of XTEA for a fixed key and round constants $\mathrm{adp}^F(k,\delta|\ da \rightarrow dd)$. *Complexity:* $O(n) < c \leq O(2^n)$.
- file common.cc

  Common functions used accross all YAARX programs.
- file eadp-tea-f-program.cc

  The probability $\mathrm{eadp}^F(da \rightarrow dd)$ *with user-provided input.*
- file eadp-tea-f.cc

  The expected additive differential probability (EADP) of the F-function of TEA, averaged over all round keys and constants: $\mathrm{eadp}^F(da \rightarrow dd)$. *Complexity:* $O(n)$.
- file max-adp-xor-fi-program.cc

  The probability ($\max_{dc} \mathrm{adp}^{\oplus}(a,db \rightarrow dc)$*) with user-provided input.*
- file max-adp-xor-fi.cc

  The maximum ADD differential probability of XOR with one fixed input-: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(a,db \rightarrow dc)$.
- file max-adp-xor-program.cc

  Maximum ADD differential probability of XOR ($\max_{dc} \mathrm{adp}^{\oplus}(da,db \rightarrow dc)$*) with user-provided input.*
- file max-adp-xor.cc

  The maximum ADD differential probability of XOR: $\max_{dc} \mathrm{adp}^{\oplus}(da,db \rightarrow dc)$.
- file max-adp-xor3-program.cc

  The probability $\max_{dd} \mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$ *with user-provided input.*
- file max-adp-xor3-set.cc

  The maximum ADD differential probability of XOR with three inputs, where one of the inputs satisfies a set of ADD differences: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da,db,\{dc_0,dc_1,\ldots\} \rightarrow dd)$.
- file max-adp-xor3.cc

  The maximum ADD differential probability of XOR with three inputs: $\max_{dd}\ \mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$.
- file max-eadp-tea-f-program.cc

  The probability $\max_{dd} \mathrm{eadp}^F(da \rightarrow dd)$ *with user-provided input.*
- file max-xdp-add-program.cc

  The probability $\max_{dc} \mathrm{xdp}^{+}(da,db \rightarrow dc)$ *with user-provided input.*
- file max-xdp-add.cc

  The maximum XOR differential probability of ADD: $\max_{dc} \mathrm{xdp}^{+}(da,db \rightarrow dc)$.
- file tea-add-ddt-search.cc

  Automatic search for ADD differential trails in TEA using full DDT-s.
- file tea-add-threshold-search.cc

  Automatic search for ADD differential trails in block cipher TEA.
- file tea-f-add-pddt.cc

  Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA.
- file tea.cc

  Common functions used in the analysis of TEA.
- file xdp-add-pddt.cc

Compute a partial difference distribution table (pDDT) for xdp^{+}.

- file xdp-add-program.cc

  Program for computing xdp^{+} *with user-provided input.*
- file xdp-add.cc

  The XOR differential probability of ADD $\mathrm{xdp}^{+}(da, db \rightarrow db)$.
- file xdp-tea-f-fk.cc

  The XOR differential probability (XDP) of the F-function of TEA for a fixed key and round constants: $\mathrm{xdp}^{F}(k_0, k_1, \delta | \, da \rightarrow dd)$.
- file xdp-xtea-f-fk.cc

  The XOR differential probability (XDP) of the F-function of XTEA for a fixed key and round constants: $\mathrm{xdp}^{F}(k, \delta | \, da \rightarrow dd)$.
- file xtea-add-threshold-search.cc

  Automatic search for ADD differential trails in block cipher XTEA.
- file xtea-f-add-pddt.cc

  Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher XTEA.
- file xtea-f-xor-pddt.cc

  Computing an XOR partial difference distribution table (pDDT) for the F-function of block cipher XTEA.
- file xtea-xor-threshold-search.cc

  Automatic search for XOR differential trails in block cipher XTEA.
- file xtea.cc

  Common functions used in the analysis of block cipher XTEA.

5.3 tests/ Directory Reference

Files

- file adp-rsh-xor-tests.cc

  Tests for adp-rsh-xor.cc.
- file adp-shift-tests.cc

  Tests for adp-shift.cc.
- file adp-tea-f-fk-ddt-tests.cc

  Tests for adp-tea-f-fk-ddt.cc.
- file adp-tea-f-fk-noshift-tests.cc

  Tests for adp-tea-f-fk-noshift.cc.
- file adp-tea-f-fk-tests.cc

  Tests for adp-tea-f-fk.cc.
- file adp-xor-fi-tests.cc

  Tests for adp-xor-fi.cc.
- file adp-xor-pddt-tests.cc

  Tests for adp-xor-pddt.cc.
- file adp-xor-tests.cc

  Tests for adp-xor.cc.

- file adp-xor3-tests.cc

  Tests for adp-xor3.cc.
- file adp-xtea-f-fk-tests.cc

  Tests for adp-xtea-f-fk.cc.
- file eadp-tea-f-tests.cc

  Tests for eadp-tea-f.cc.
- file max-adp-xor-fi-tests.cc

  Tests for max-adp-xor-fi.cc.
- file max-adp-xor-tests.cc

  Tests for max-adp-xor.cc.
- file max-adp-xor3-set-tests.cc

  Tests for max-adp-xor3-set.cc.
- file max-adp-xor3-tests.cc

  Tests for max-adp-xor3.cc.
- file max-xdp-add-tests.cc

  Tests for max-xdp-add.cc.
- file tea-add-ddt-search-tests.cc

  Tests for tea-add-ddt-search.cc.
- file tea-add-threshold-search-tests.cc

  Tests for tea-add-threshold-search.cc.
- file tea-f-add-pddt-tests.cc

  Tests for tea-f-add-pddt.cc.
- file **xdp-add-pddt-tests.cc**
- file xdp-add-tests.cc

  Tests for xdp-add.cc.
- file xdp-tea-f-fk-tests.cc

  Tests for xdp-tea-f-fk.cc.
- file xdp-xtea-f-fk-tests.cc

  Tests for xdp-xtea-f-fk.cc.
- file xtea-add-threshold-search-tests.cc

  Tests for xtea-add-threshold-search.cc.
- file xtea-xor-threshold-search-tests.cc

  Tests for xtea-xor-threshold-search.cc.

6 Data Structure Documentation

6.1 difference_t Struct Reference

```
#include <common.hh>
```

Data Fields

- uint32_t dx
- double p

6.1.1 Detailed Description

A difference structure.

6.1.2 Field Documentation

6.1.2.1 uint32_t difference_t::dx

A difference.

6.1.2.2 double difference_t::p

Probability with which dx holds.

The documentation for this struct was generated from the following file:

- include/common.hh

6.2 differential_3d_t Struct Reference

```
#include <common.hh>
```

Data Fields

- uint32_t da
- uint32_t db
- uint32_t dc
- double p

6.2.1 Detailed Description

A differential composed of three differences. For example, da and db can be input differences to XOR and dc can be the corresponding output difference. The differential holds with probability p.

6.2.2 Field Documentation

6.2.2.1 uint32_t differential_3d_t::da

Input difference.

6.2.2.2 uint32_t differential_3d_t::db

Input difference.

6.2.2.3 uint32_t differential_3d_t::dc

Output difference.

6.2.2.4 double differential_3d_t::p

Probability of the differential.

The documentation for this struct was generated from the following file:

- include/common.hh

6.3 differential_t Struct Reference

```
#include <common.hh>
```

Data Fields

- uint32_t dx
- uint32_t dy
- uint32_t npairs
- double p

6.3.1 Detailed Description

A differential composed of two differences.

6.3.2 Field Documentation

6.3.2.1 uint32_t differential_t::dx

Input difference.

6.3.2.2 uint32_t differential_t::dy

Input difference.

6.3.2.3 uint32_t differential_t::npairs

Number of right pairs.

6.3.2.4 double differential_t::p

Probability of the differential.

The documentation for this struct was generated from the following file:

- include/common.hh

6.4 skey_t Struct Reference

Data Fields

- uint32_t **k0**
- uint32_t **k1**
- double **p**

6.4.1 Detailed Description

The key-dependent probability for the TEA F-function with the corresponding value of the round keys.

The documentation for this struct was generated from the following file:

- tests/adp-tea-f-fk-ddt-tests.cc

6.5 struct_comp_diff_3d_p Struct Reference

```
#include <common.hh>
```

Public Member Functions

- bool **operator()** (differential_3d_t a, differential_3d_t b) const

6.5.1 Detailed Description

Comparing 3d differentials by probability.

The documentation for this struct was generated from the following file:

- include/common.hh

6.6 struct_comp_diff_dx_dy Struct Reference

```
#include <common.hh>
```

Public Member Functions

- bool **operator()** (differential_t a, differential_t b)

6.6.1 Detailed Description

Compare two differentials a,b by the magnitute of the indexes a_idx, b_idx: lower indices are listed first. For example, the indices of the differentials a(dx,dy,p) and b(dx,dy,p) are a_idx = (a.dx 2^{n} + a.dy) = (a.dx | a.dy) and b_idx = (b.dx 2^{n} + b.dy) = (b.dx | b.dy) where n is the word size and '|' denotes concatenation. Thus a_idx and b_idx are compared.

The documentation for this struct was generated from the following file:

- include/common.hh

6.7 struct_comp_diff_p Struct Reference

```
#include <common.hh>
```

Public Member Functions

- bool **operator()** (differential_t a, differential_t b) const

6.7.1 Detailed Description

Comparing 2d differentials by probability.

The documentation for this struct was generated from the following file:

- include/common.hh

7 File Documentation

7.1 include/adp-rsh-xor.hh File Reference

Header file for adp-rsh-xor.cc: The ADD differential probability of right shift followed by XOR: $\mathrm{adp}^{\gg\oplus}$. .

Defines

- #define ADP_RSH_XOR_NSTATES 3
- #define ADP_RSH_XOR_NPOS 3
- #define ADP_RSH_XOR_MSIZE (1L << ADP_RSH_XOR_NSTATES)
- #define ADP_RSH_XOR_NINPUTS 2
- #define ADP_RSH_XOR_NOUTPUTS 1
- #define ADP_RSH_XOR_COLSUM 4
- #define ADP_RSH_XOR_NORM 1.0 /(double)ADP_RSH_XOR_COLSUM

Functions

- uint32_t rsh_xor (uint32_t a, uint32_t x, int r)
- double adp_rsh_xor_exper (const uint32_t da, const uint32_t dx, const uint32_t db, const int r)
- void adp_rsh_xor_alloc_matrices (gsl_matrix *A[3][2][2][2])
- void adp_rsh_xor_free_matrices (gsl_matrix *A[3][2][2][2])
- void adp_rsh_xor_normalize_matrices (gsl_matrix *A[3][2][2][2])
- void adp_rsh_xor_print_matrices (gsl_matrix *A[3][2][2][2])
- void adp_rsh_xor_sf (gsl_matrix *A[3][2][2][2])
- double adp_rsh_xor (gsl_matrix *A[3][2][2][2], uint32_t da, uint32_t dx, uint32_t db, int r)
- double adp_rsh_xor_approx (uint32_t da, uint32_t dx, uint32_t db, int r)

7.1.1 Detailed Description

Header file for adp-rsh-xor.cc: The ADD differential probability of right shift followed by XOR: $\mathrm{adp}^{\gg\oplus}$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.1.2 Define Documentation

7.1.2.1 #define ADP_RSH_XOR_COLSUM 4

Sum of the non-zero elements in one column of the $\mathrm{adp}^{\gg\oplus}$ matrices.

7.1.2.2 #define ADP_RSH_XOR_MSIZE (1L << ADP_RSH_XOR_NSTATES)

Size of the transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

7.1.2.3 #define ADP_RSH_XOR_NINPUTS 2

Number of inputs to the operation $(\gg\oplus)$.

7.1.2.4 #define ADP_RSH_XOR_NORM 1.0 /(double)ADP_RSH_XOR_COLSUM

Normalization factor for transforming the elements of the matrices into probabilities.

7.1.2.5 #define ADP_RSH_XOR_NOUTPUTS 1

Number of outputs from the operation $(\gg\oplus)$.

7.1.2.6 #define ADP_RSH_XOR_NPOS 3

Special bit positions in the computation of $\mathrm{adp}^{\gg\oplus}$.

7.1.2.7 #define ADP_RSH_XOR_NSTATES 3

Number of states of the S-function for $\mathrm{adp}^{\gg\oplus}$.

7.1.3 Function Documentation

7.1.3.1 double adp_rsh_xor (gsl_matrix * *A[3][2][2][2]*, uint32_t *da*, uint32_t *dx*, uint32_t *db*, int *r*)

The ADD differential probability of $(\gg\oplus)$ (RSH-XOR) computed experimentally over all inputs. Complexity: $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.
da	input difference.
dx	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\gg\oplus}(r|da,dx \rightarrow db)$.

See also

adp_rsh_xor_exper

7.1.3.2 void adp_rsh_xor_alloc_matrices (gsl_matrix * *A[3][2][2][2]*)

Allocate memory for the transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

See also

adp_rsh_xor_free_matrices

7.1.3.3 double adp_rsh_xor_approx (uint32_t *da*, uint32_t *dx*, uint32_t *db*, int *r*)

Approximation of $\mathrm{adp}^{\gg\oplus}$ obtained as the multiplication of the differential probabilities $\mathrm{adp}^{\gg}$ and $\mathrm{adp}^{\oplus}$.

Parameters

da	input difference.
dx	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\gg\oplus}(r|da,dx \rightarrow db) \approx \mathrm{adp}^{\gg} \cdot \mathrm{adp}^{\oplus}$.

See also

adp_xor, adp_rsh

7.1.3.4 double adp_rsh_xor_exper (const uint32_t *da*, const uint32_t *dx*, const uint32_t *db*, const int *r*)

The ADD differential probability of RSH-XOR computed experimentally over all inputs. Complexity: $O(2^{2n})$.

Parameters

da	input difference.
dx	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\gg\oplus}(r|da,dx \rightarrow db)$.

See also

adp_rsh_xor

7.1.3.5 void adp_rsh_xor_free_matrices (gsl_matrix ∗ *A[3][2][2][2]*)

Free memory reserved for the transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

See also

adp_rsh_xor_alloc_matrices

7.1.3.6 void adp_rsh_xor_normalize_matrices (gsl_matrix * A[3][2][2][2])

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

7.1.3.7 void adp_rsh_xor_print_matrices (gsl_matrix * A[3][2][2][2])

Print the elements of A.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

7.1.3.8 void adp_rsh_xor_sf (gsl_matrix * A[3][2][2][2])

S-function for the operation $(\gg \oplus)$ (RSH-XOR).

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\mathrm{adp}^{\gg\oplus}$.

$A[3][2][2][2] = A[j][da[i]][dx[i+r]][db[i]]$, where $da[i]$ denotes the i-th bit of da, n is the word size, r is the shift constant, i is the bit position and j is a special bit position with three possible values:

- $j = 0 : 0 \leq i < n - r$.
- $j = 1 : n - r < i < n$.
- $j = 2 : i = n - r$.

7.1.3.9 uint32_t rsh_xor (uint32_t *a*, uint32_t *x*, int *r*)

The sequence of operations right shift (RSH) followed by an XOR (RSH-XOR).

Parameters

a	input to XOR.
x	input to RSH.
r	shift constant.

Returns

$b = a \oplus (x \gg r)$.

7.2 include/adp-shift.hh File Reference

Header file for adp-shift.cc: The ADD differential probability of left shift (LSH): $\mathrm{adp}^{\ll}$ and right shift (RSH): $\mathrm{adp}^{\gg}$. .

Functions

- double adp_lsh_exper (uint32_t da, uint32_t db, int l)
- double adp_lsh (uint32_t da, uint32_t db, int l)
- double adp_rsh_exper (const uint32_t da, const uint32_t db, const int r)
- double adp_rsh (uint32_t da, uint32_t db, int r)
- void adp_rsh_odiffs (uint32_t dx[4], const uint32_t da, int r)

7.2.1 Detailed Description

Header file for adp-shift.cc: The ADD differential probability of left shift (LSH): $\mathrm{adp}^{\ll}$ and right shift (RSH): $\mathrm{adp}^{\gg}$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.2.2 Function Documentation

7.2.2.1 double adp_lsh (uint32_t *da*, uint32_t *db*, int *l*)

The ADD differential probability of $(\ll)$ (LSH). Complexity: $O(1)$.

Parameters

da	input difference.
db	output difference.
l	shift constant.

Returns

$\mathrm{adp}^{\ll}(l|\ da \rightarrow db)$.

See also

adp_lsh_exper

7.2.2.2 double adp_lsh_exper (uint32_t *da*, uint32_t *db*, int *l*)

The ADD differential probability of $(\lll)$ (LSH) computed experimentally over all inputs. Complexity: $O(2^n)$.

Parameters

da	input difference.
db	output difference.
l	shift constant.

Returns

$\mathrm{adp}^{\lll}(l|\ da \rightarrow db)$.

See also

adp_lsh

7.2.2.3 double adp_rsh (uint32_t *da*, uint32_t *db*, int *r*)

The ADD differential probability of $(\ggg)$ (RSH). Complexity: $O(1)$.

Parameters

da	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\ggg}(r|\ da \rightarrow db)$.

See also

adp_rsh_exper

Note

$db \in \{(da \ggg 5), (da \ggg 5)+1, (da \ggg 5)-2^{n-5}, (da \ggg 5)-2^{n-5}+1\}$.

7.2.2.4 double adp_rsh_exper (const uint32_t *da*, const uint32_t *db*, const int *r*)

The ADD differential probability of $(\ggg)$ (RSH) computed experimentally over all inputs. Complexity: $O(2^n)$.

Parameters

da	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\gg}(r|\ da \rightarrow db)$.

See also

adp_rsh

7.2.2.5 void adp_rsh_odiffs (uint32_t *dx[4]*, const uint32_t *da*, int *r*)

Compute the set of possible output differences dx after a right shift by r

Parameters

da	input difference.
r	shift constant.
dx	the set of all 4 possible output differences.

7.3 include/adp-tea-f-fk-ddt.hh File Reference

Header file for adp-tea-f-fk-ddt.cc: Computing the full difference distribution table (DDT) for the F-function of block cipher TEA by exaustive search over all inputs. Complexity $O(2^{2n})$. .

Functions

- void ddt_sort_rows (differential_t **T)
- bool comp_rows (differential_t *a, differential_t *b)
- void ddt_sort_first_col (differential_t **T)
- void ddt_to_list (uint32_t **DDT, differential_t *SDDT)
- void ddt_to_diff_struct (uint32_t **DDT, differential_t **SDDT)
- void ddt_sort (differential_t *SDDT)
- void print_rsddt (differential_t **RSDDT)
- void print_sddt (differential_t *SDDT)
- double adp_f_exper_fixed_key_all (const uint32_t da, const uint32_t db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_adp_f_exper_fixed_key_all (const uint32_t da, uint32_t *db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- differential_t ** rsddt_alloc ()

- void rsddt_free (differential_t **T)
- differential_t * sddt_alloc ()
- void sddt_free (differential_t *ST)
- uint32_t ** ddt_alloc ()
- void ddt_free (uint32_t **T)
- void ddt_f (uint32_t **T, uint32_t k0, uint32_t k1, uint32_t delta, uint32_t lsh_-const, uint32_t rsh_const)
- void ddt_print (uint32_t **T)
- double adp_f_ddt (uint32_t **DDT, uint32_t dx, uint32_t dy)
- double max_adp_f_ddt (uint32_t **DDT, uint32_t dx, uint32_t *dy)
- double max_adp_f_rsddt (differential_t **TS, uint32_t dx, uint32_t *dy)
- uint32_t *** xddt_alloc ()
- void xddt_free (uint32_t ***T)
- differential_t *** xrsddt_alloc ()
- void xrsddt_free (differential_t ***T)
- differential_t ** xsddt_alloc ()
- void xsddt_free (differential_t **ST)

7.3.1 Detailed Description

Header file for adp-tea-f-fk-ddt.cc: Computing the full difference distribution table (DDT) for the F-function of block cipher TEA by exaustive search over all inputs. Complexity $O(2^{2n})$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.3.2 Function Documentation

7.3.2.1 double adp_f_ddt (uint32_t ** *DDT*, uint32_t *dx*, uint32_t *dy*)

Compute the ADD differential probability of the TEA F-function from the full DDT, pre-computed for a a fixed key and round constant.

Parameters

DDT	DDT.
dx	input difference.
dy	output difference.

Returns

DDT[da][db] = $\mathrm{adp}^F(k_0, k_1, \delta | \ da \to dd)$.

See also

adp_f_exper_fixed_key_all

7.3.2.2 double adp_f_exper_fixed_key_all (const uint32_t *da*, const uint32_t *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the ADD differential probability of the TEA F-function for a fixed key and round constants ($\mathrm{adp}^F(k_0, k_1, \delta | \ da \to dd)$) by exhaustive searc over all inputs. Complexity $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta | \ da \to dd)$.

7.3.2.3 bool comp_rows (differential_t * *a*, differential_t * *b*)

Compare two rows in a row-sorted DDT by their first (max) element. Assumes that the elemnets in a row are sorted in descending order.

Parameters

a	row of differentials in a DDT.
b	row of differentials in a DDT.

7.3.2.4 uint32_t** ddt_alloc ( )

Allocate memory for a DDT as a 2D array containingg number of rigt pairs.

Returns

a DDT as a 2D array containing number of rigt pairs.

See also

ddt_free

7.3.2.5 void ddt_f (uint32_t ** *T*, uint32_t *k0*, uint32_t *k1*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the full difference distribution table (DDT) for the F-function of block cipher TEA for a fixed key and round constant, by exaustive search over all input values and differences. Complexity $O(2^{2n})$.

Parameters

T	a DDT as a 2D array containing number of right pairs..
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

full DDT for the TEA F-function.

7.3.2.6 void ddt_free (uint32_t ** *T*)

Free the memory reserved for a DDT as a 2D array containingg number of rigt pairs.

Parameters

T	a DDT as a 2D array containing number of rigt pairs.

See also

ddt_alloc

7.3.2.7 void ddt_print (uint32_t ** *T*)

Print the entries of a DDT.

Parameters

T	DDT.

7.3.2.8 void ddt_sort (differential_t * *SDDT*)

Sort all elements of a DDT, represented as a 1D list of differentials, in descending order by the number of right pairs.

Parameters

SDDT	DDT as a 1D list of differentials.

7.3.2.9 void ddt_sort_first_col (differential_t $**$ *T*)

Sorts the rows of a difference distribution table (DDT) 2D by the probability of the elements in the first column -- highest probability first.

Parameters

T	a difference distribution table (DDT).

7.3.2.10 void ddt_sort_rows (differential_t $**$ *T*)

Sort every row by decreasing number of right pairs.

Parameters

T	a difference distribution table (DDT).

7.3.2.11 void ddt_to_diff_struct (uint32_t $**$ *DDT,* differential_t $**$ *SDDT*)

Convert a DDT to 2D array of differentials.

Parameters

DDT	difference distribution table.
SDDT	array differentials.

7.3.2.12 void ddt_to_list (uint32_t $**$ *DDT,* differential_t $*$ *SDDT*)

Convert a DDT to a list of differentials.

Parameters

DDT	difference distribution table.
SDDT	list of differentials.

7.3.2.13 double max_adp_f_ddt (uint32_t $**$ *DDT,* uint32_t *dx,* uint32_t $*$ *dy*)

For a fixed input difference to the TEA F-function compute the maximum probability output ADD difference from the full DDT, precomputed for a fixed key and round constant. Complexity $O(1)$.

Parameters

DDT	DDT.
dx	input difference.
dy	maximum probability output difference.

Returns

$\max_{dd} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \to dy)$.

See also

max_adp_f_exper_fixed_key_all

7.3.2.14 double max_adp_f_exper_fixed_key_all (const uint32_t *da*, uint32_t * *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For a fixed input difference to the TEA F-function compute the maximum probability output ADD difference for a fixed key and round constant by exhaustive search over all inputs and output differences. Complexity $O(2^{2n})$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dd} \mathrm{adp}^F(k_0, k_1, \delta | \, da \to dd)$.

See also

max_adp_f_ddt, max_adp_f_rsddt

7.3.2.15 double max_adp_f_rsddt (differential_t ** *TS*, uint32_t *dx*, uint32_t * *dy*)

For a fixed input difference to the TEA F-function compute the maximum probability output ADD difference from the full DDT represented as a 2D array of differentials. In this DDT the differentials in every row are sorted by decreasing probability. Complexity $O(1)$.

Parameters

TS	a DDT in which the differentials in every row are sorted by decreasing number of probability.
dx	input difference.
dy	maximum probability output difference.

Returns

TS[dx][0] = $\max_{dd} \mathrm{adp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$.

See also

max_adp_f_ddt, max_adp_f_exper_fixed_key_all

7.3.2.16 void print_rsddt (differential_t ∗∗ *RSDDT*)

Print the elements of a DDT.

Parameters

RSDDT	DDT as a 2D list of differentials.

7.3.2.17 void print_sddt (differential_t ∗ *SDDT*)

Print the elements of a DDT.

Parameters

SDDT	DDT as a 1D list of differentials.

7.3.2.18 differential_t∗∗ rsddt_alloc ( )

Allocate memory for a DDT as a 2D array of differentials.

Returns

a DDT as a 2D array of differentials.

See also

rsddt_free

7.3.2.19 void rsddt_free (differential_t ∗∗ *T*)

Free the memory reserved for a DDT as a 2D array of differentials.

Parameters

T	a DDT as a 2D array of differentials.

See also

rsddt_alloc

7.3.2.20 differential_t* sddt_alloc ( )

Allocate memory for a DDT as a 1D array of differentials.

Returns

a DDT as a 1D array of differentials.

See also

sddt_free

7.3.2.21 void sddt_free (differential_t * *ST*)

Free the memory reserved for a DDT as a 1D array of differentials.

Parameters

ST	a DDT as a 1D array of differentials.

See also

sddt_alloc

7.3.2.22 uint32_t*** xddt_alloc ( )

Allocate memory for a an array of NDELTA DDTs. Each DDT represents a 2D array containing numbers of rigt right pairs and generated for a fixed value of the δ constant of the TEA F-function.

Returns

array of DDTs: each DDT represents a 2D array containing number of right pairs.

7.3.2.23 void xddt_free (uint32_t *** *T*)

Free the memory reserved from a previous call to xddt_alloc()

Parameters

T	an array of DDTs: each DDT is a 2D array containing number of right pairs.

7.3.2.24 differential_t∗∗∗ xrsddt_alloc ( )

Allocate memory for a an array of NDELTA DDTs. Each DDT represents a 2D array of differentials, generated for a fixed value of the δ constant of the TEA F-function.

Returns

array of DDTs: each DDT represents a 2D array of differentials.

7.3.2.25 void xrsddt_free (differential_t ∗∗∗ *T*)

Free the memory reserved from a previous call to xrsddt_alloc()

Parameters

T	an array of DDTs: each DDT is a 2D array of differentials.

7.3.2.26 differential_t∗∗ xsddt_alloc ( )

Allocate memory for a an array of NDELTA DDTs. Each DDT represents a 1D list of differentials, generated for a fixed value of the δ constant of the TEA F-function.

Returns

array of DDTs: each DDT represents a 1D list of differentials,

7.3.2.27 void xsddt_free (differential_t ∗∗ *ST*)

Free the memory reserved from a previous call to xrsddt_free()

Parameters

ST	an array of DDTs: each DDT is a 1D list of differentials.

7.4 include/adp-tea-f-fk-noshift.hh File Reference

Header file for adp-tea-f-fk-noshift.cc: The additive differential probability (ADP) of a modified version of the F-function of TEA with the shift operations removed. Complexity $O(n)$. .

Defines

- #define ADP_F_OP_NOSHIFT_NINPUTS 4
- #define ADP_F_OP_NOSHIFT_MSIZE (1L $<<$ 7)
- #define ADP_F_OP_NOSHIFT_NMATRIX 32
- #define ADP_F_OP_NOSHIFT_COLSUM 2
- #define ADP_F_OP_NOSHIFT_NORM 1.0 /(double)ADP_F_OP_NOSHIFT_COLSUM

- #define ADP_F_OP_NOSHIFT_ISTATE 64
- #define NSPOS 1

Functions

- void adp_f_op_noshift_sf (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_alloc_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_free_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_normalize_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_print_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- double adp_f_op_noshift (gsl_matrix ∗A[NSPOS][2][2][2][2][2], uint32_t k0, uint32_t k1, uint32_t delta, uint32_t da, uint32_t db)
- double adp_f_op_noshift_exper (uint32_t k0, uint32_t k1, uint32_t delta, uint32_t da, uint32_t db)

7.4.1 Detailed Description

Header file for adp-tea-f-fk-noshift.cc: The additive differential probability (ADP) of a modified version of the F-function of TEA with the shift operations removed. Complexity $O(n)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.4.2 Define Documentation

7.4.2.1 #define ADP_F_OP_NOSHIFT_COLSUM 2

Sum of the non-zero elements in one column of the F' matrices.

7.4.2.2 #define ADP_F_OP_NOSHIFT_ISTATE 64

Initial state for start of the compuation of the ADP of F'.

7.4.2.3 #define ADP_F_OP_NOSHIFT_MSIZE (1L $<<$ 7)

Number of states of the S-function for F'.

7.4.2.4 #define ADP_F_OP_NOSHIFT_NINPUTS 4

Number of inputs to F': k_0, k_1, δ, da.

7.4.2.5 #define ADP_F_OP_NOSHIFT_NMATRIX 32

Number of transition probability matrices for F'.

7.4.2.6 #define ADP_F_OP_NOSHIFT_NORM 1.0 /(double)ADP_F_OP_NOSHIFT_COLSUM

Normalization factor for transforming the elements of the matrices into probabilities.

7.4.2.7 #define NSPOS 1

Number of special positions for ADP of F'.

7.4.3 Function Documentation

7.4.3.1 double adp_f_op_noshift (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2],* uint32_t *k0,* uint32_t *k1,* uint32_t *delta,* uint32_t *da,* uint32_t *db*)

The additive differential probability (ADP) of a modified version of the F-function of TEA with the shift operations removed, denoted by F' and defined as:

$y = F'(k_0, k_1, \delta | x) = (x + k_0) \oplus (x + \delta) \oplus (x + k_1)$.

Complexity: $O(n)$.

Parameters

A	transition probability matrices for F' computed with adp_f_op_noshift_sf
k0	first round key.
k1	second round key.
delta	round constant.
da	input difference.
db	output difference.

Returns

$\mathrm{adp}^{F'}(k_0, k_1, \delta | da \rightarrow db)$.

7.4.3.2 void adp_f_op_noshift_alloc_matrices (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2]*)

Allocate memory for the transition probability matrices for F'.

Parameters

A	transition probability matrices for F'.

See also

adp_rsh_xor_free_matrices

7.4.3.3 double adp_f_op_noshift_exper (uint32_t *k0*, uint32_t *k1*, uint32_t *delta*, uint32_t *da*, uint32_t *db*)

The additive differential probability (ADP) of F' (a modified version of the F-function of TEA with the shift operations removed) computed experimentally over all inputs. - Complexity: $O(2^{2n})$.

Parameters

k0	first round key.
k1	second round key.
delta	round constant.
da	input difference.
db	output difference.

Returns

$\mathrm{adp}^{F'}(k_0, k_1, \delta | \, da \rightarrow db)$.

See also

adp_f_op_noshift

7.4.3.4 void adp_f_op_noshift_free_matrices (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2]*)

Free memory reserved by a previous call to adp_rsh_xor_free_matrices.

Parameters

A	transition probability matrices for F'.

7.4.3.5 void adp_f_op_noshift_normalize_matrices (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for F'.

7.4.3.6 void adp_f_op_noshift_print_matrices (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2]*)

Print the elements of A.

Parameters

A	transition probability matrices for F'.

7.4.3.7 void adp_f_op_noshift_sf (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2]*)

S-function for a modified version of the TEA F-function with the shift operations removed, denoted by F' and defined as:

$y = F'(k_0, k_1, \delta | \, x) = (x + k_0) \oplus (x + \delta) \oplus (x + k_1)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for F'.

$A[j][2][2][2][2][2] = A[j][k0[i]][k1[i]][\delta[i]][da[i]][db[i]]$, where

- j : dummy variable for future use.
- $k_0[i]$: the i-th bit of the first round key.
- $k_1[i]$: the i-th bit of the second round key.
- $\delta[i]$: the i-th bit of the round constant:
- $da[i]$: the i-th bit of the input difference.
- $db[i]$: the i-th bit of the output difference.

7.5 include/adp-tea-f-fk.hh File Reference

Header file for adp-tea-f-fk.cc: The ADD differential probability of the F-function of TEA for a fixed key and round constants $\mathrm{adp}^F(k_0, k_1, \delta | \, da \rightarrow dd)$, where $F(k_0, k_1, \delta | \, x) = ((x \ll 4) + k_0) \oplus (x + \delta) \oplus ((x \gg 5) + k_1)$ Complexity: $O(n) < c \leq O(2^n)$. .

Functions

- bool adp_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool adp_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32-_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, int32_t x)
- uint32_t adp_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_-t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double adp_f_fk (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)

- uint32_t adp_f_assign_bit_x_dx (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t *x_cnt, double *ret_prob, uint32_t *ret_dx)
- double max_dx_adp_f_fk (const uint32_t n, uint32_t *ret_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint32_t adp_f_assign_bit_x_dy (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t *x_cnt, double *ret_prob, uint32_t *ret_dy)
- double max_dy_adp_f_fk (const uint32_t n, const uint32_t dx, uint32_t *ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double all_dy_adp_f_fk (const uint32_t n, const uint32_t dx, uint32_t *ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, uint64_t *x_cnt)
- uint32_t adp_f_assign_bit_x_dx_dy (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, differential_t *x_cnt, double *ret_prob, uint32_t *ret_dx, uint32_t *ret_dy)
- double max_dx_dy_adp_f_fk (const uint32_t n, uint32_t *ret_dx, uint32_t *ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint32_t adp_f_assign_bit_x_dx_key (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ***x_cnt, double *ret_prob, uint32_t *ret_dx, uint32_t *ret_k0, uint32_t *ret_k1)
- double max_key_dx_adp_f_fk (const uint32_t n, uint32_t *ret_dx, const uint32_t dy, uint32_t *ret_k0, uint32_t *ret_k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double adp_f_fk_v2 (const uint32_t da, const uint32_t dd, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- void f_sfun (const uint32_t n, const uint32_t x_word, const uint32_t dx_word, const uint32_t delta_word, const uint32_t k0_word, const uint32_t k1_word)
- double adp_f_fk_exper (const uint32_t da, const uint32_t db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_dx_adp_f_fk_exper (uint32_t *max_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_dy_adp_f_fk_exper (const uint32_t dx, uint32_t *max_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_dx_dy_adp_f_fk_exper (uint32_t *max_dx, uint32_t *max_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)

- uint64_t ∗∗∗ x_cnt_alloc ()
- void x_cnt_free (uint64_t ∗∗∗x_cnt)
- void x_cnt_print (uint32_t ∗∗∗x_cnt)

7.5.1 Detailed Description

Header file for adp-tea-f-fk.cc: The ADD differential probability of the F-function of TEA for a fixed key and round constants $\mathrm{adp}^F(k_0, k_1, \delta | \, da \rightarrow dd)$, where $F(k_0, k_1, \delta | \, x) = ((x \ll 4) + k_0) \oplus (x + \delta) \oplus ((x \gg 5) + k_1)$ Complexity: $O(n) < c \leq O(2^n)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.5.2 Function Documentation

7.5.2.1 uint32_t **adp_f_assign_bit_x** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint32_t ∗ *x_cnt*, double ∗ *prob*)

Counts the number of values x for which the differential $(dx \rightarrow dy)$ for the F-function of TEA is satisfied. The function operates by recursively assigning the bits of x starting from bit position i and terminating at the MS bit n. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the i LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the i LS bits of x.
x	input value of size at least (i + rsh_const).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key ADD probability of F: $\mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_f_fk

7.5.2.2 uint32_t adp_f_assign_bit_x_dx (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dx*)

For given output difference `dy`, compute all input differences `dx` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability input difference.

The function works by recursively assigning the bits of `x` and `dx` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat .

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dx`.
ret_prob	the maximum probability over all input differences $\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dx	the input difference that has maximum probability.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_f_assign_bit_x, max_dx_adp_f_fk

7.5.2.3 uint32_t adp_f_assign_bit_x_dx_dy (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, differential_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dx*, uint32_t * *ret_dy*)

For the TEA F-functuion with fixed key and round constant, compute all differentials $(dx \rightarrow dy)$ and their probabilities.

The function works by recursively assigning the bits of `x`, `dx` and `dy` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^{2n} differentials $(dx \rightarrow dy)$ and their probabilities.
ret_prob	the maximum probability over all input and output differences $\max_{dy,dy} \mathrm{adp}^F(k_0, k_1, \delta \vert\, dx \rightarrow dy)$.
ret_dx	the input difference of the maximum probability differential.
ret_dy	the output difference of the maximum probability differential.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_assign_bit_x_dx, adp_f_assign_bit_x_dy.

7.5.2.4 uint32_t adp_f_assign_bit_x_dx_key (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t *** *x_cnt*, double * *ret_prob*, uint32_t * *ret_dx*, uint32_t * *ret_k0*, uint32_t * *ret_k1*)

For the TEA F-function with fixed round constant, and for a fixed output difference `dy`, compute all differentials $(dx \rightarrow dy)$ and their probabilities for all values of the round keys `k0`, `k1`.

The function works by recursively assigning the bits of x, dx, k0 and k1 starting at bit position i and terminating at the MS bit n. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the i LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the i LS bits of x.
x	input value of size at least (i + rsh_const).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^{3n} differentials $(dx \rightarrow dy)$ and their probabilities.
ret_prob	the maximum probability over all input differences and round keys $\max_{dx,k_0,k_1} \mathrm{adp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dx	the input difference of the maximum probability differential.
ret_k0	the first round key for the maximum probability differential.
ret_k1	the second round key for the maximum probability differential.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_f_assign_bit_x_dx_key, adp_f_assign_bit_x_dx_dy.

7.5.2.5 uint32_t adp_f_assign_bit_x_dy (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dy*)

For given input difference dx, compute all output differences dy and their probabilities, by counting all values x that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability output difference.

The function works by recursively assigning the bits of x and dy starting at bit position i and terminating at the MS bit n. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the i LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the i LS bits of x.

x	input value of size at least (`i + rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dy`.
ret_prob	the maximum probability over all output differences $\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta \vert\, dx \rightarrow dy)$.
ret_dy	the output difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_assign_bit_x_dx

7.5.2.6 bool adp_f_check_x (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value `x` satisfies the ADD differential $(dx \rightarrow dy)$ for the TEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k_0, k_1, \delta : dy = F(x+dx) - F(x)$.

7.5.2.7 double adp_f_fk (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher TEA: $\mathrm{adp}^F(k_0, k_1, \delta |\ dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta |\ dx \rightarrow dy)$.

See also

adp_f_assign_bit_x

7.5.2.8 double adp_f_fk_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher TEA: $\mathrm{adp}^F(k_0, k_1, \delta |\ dx \rightarrow dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

adp_f_fk, adp_f_fk_v2.

7.5.2.9 double adp_f_fk_v2 (const uint32_t *da*, const uint32_t *dd*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher TEA: $\mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

The function works by dividing the input to F into independent parts and iterating over the values in each part. The resulting complexity is equivalent to exhaustive search over all inputs: $O(2^n)$.

Parameters

da	input difference.
dd	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

adp_f_fk

7.5.2.10 bool adp_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, int32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` is satisfied on the `i` LS bits of `x` i.e. check if $k_0, k_1, \delta : \; dy[i-1:0] = F(x[i-1:0] + dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i + rsh_const`).

Returns

TRUE if $k_0, k_1, \delta :\ dy[i-1:0] = F(x[i-1:0] + dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

7.5.2.11 double all_dy_adp_f_fk (const uint32_t *n*, const uint32_t *dx*, uint32_t ∗ *ret_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, uint64_t ∗ *x_cnt*)

For given input difference `dx`, compute all output differences `dy` for the TEA F-function with fixed keys and round constants. Returns the maximum output probability.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
x_cnt	array of 2^n counters - each one keeps track of the number of inputs `x` satisfying $(dx \rightarrow dy)$ for every `dy`.

Returns

$\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta |\ dx \rightarrow dy)$.

See also

max_dy_adp_f_fk

7.5.2.12 void f_sfun (const uint32_t *n*, const uint32_t *x_word*, const uint32_t *dx_word*, const uint32_t *delta_word*, const uint32_t *k0_word*, const uint32_t *k1_word*)

Compute the S-function for the TEA F function.

Parameters

n	word size.
x_word	input to F.
dx_word	input difference.
delta_word	round constant.
k0_word	first round key.
k1_word	second round key.

7.5.2.13 double max_dx_adp_f_fk (const uint32_t *n*, uint32_t * *ret_dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
ret_dx	maximum probability input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dx

7.5.2.14 double max_dx_adp_f_fk_exper (uint32_t * *max_dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

max_dx	maximum probability input difference.
dy	output difference.
k0	first round key.

k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

max_dx_adp_f_fk

7.5.2.15 double max_dx_dy_adp_f_fk (const uint32_t *n*, uint32_t ∗ *ret_dx*, uint32_t ∗ *ret_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For the TEA F-functuion with fixed key and round constant, compute the maximum probability differential $(dx \rightarrow dy)$ over all input and output differences. **Complexity:** $O(3n) < c \leq O(2^{3n})$. **Memory:** $12 \cdot 2^{2n}$ Bytes.

Parameters

n	word size.
ret_dx	the input difference of the maximum probability differential.
ret_dy	the output difference of the maximum probability differential.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx,dy} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dx_dy

7.5.2.16 double max_dx_dy_adp_f_fk_exper (uint32_t ∗ *max_dx*, uint32_t ∗ *max_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For the TEA F-functuion with fixed key and round constant, compute the maximum probability differential $(dx \rightarrow dy)$ over all input and output differences. **Complexity:** $O(2^{3n})$.

Parameters

max_dx	the input difference of the maximum probability differential.
max_dy	the output difference of the maximum probability differential.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx,dy} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \to dy)$.

See also

max_dx_dy_adp_f_fk

7.5.2.17 double **max_dy_adp_f_fk** (const uint32_t *n*, const uint32_t *dx*, uint32_t * *ret_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \to dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory requirement:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \to dy)$.

See also

adp_f_assign_bit_x_dy, max_dy_adp_f_fk

7.5.2.18 double max_dy_adp_f_fk_exper (const uint32_t *dx*, uint32_t * *max_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For given input difference dx, compute the maximum probability output difference dy over all output differences: $\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \to dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx	input difference.
max_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \to dy)$.

See also

max_dy_adp_f_fk

7.5.2.19 double max_key_dx_adp_f_fk (const uint32_t *n*, uint32_t * *ret_dx*, const uint32_t *dy*, uint32_t * *ret_k0*, uint32_t * *ret_k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For the TEA F-functuion with fixed key and round constant, compute the maximum probability differential $(dx \to dy)$ over all input differences and round keys. **Complexity:** $O(4n) < c \leq O(2^{4n})$. **Memory:** $12 \cdot 2^{3n}$ Bytes.

Parameters

n	word size.
dy	output difference.
ret_dx	the input difference of the maximum probability differential.
ret_k0	the first round key for the maximum probability differential.
ret_k1	the second round key for the maximum probability differential.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx,k_0,k_1} \mathrm{adp}^F(k_0,k_1,\delta|\, dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dx_key

7.5.2.20 uint64_t∗∗∗ x_cnt_alloc ( )

Allocate memory for a 2D array of differentials.

Returns

2D array of differentials.

See also

x_cnt_free

7.5.2.21 void x_cnt_free (uint64_t ∗∗∗ *x_cnt*)

Free the memory allocated from a previous call to x_cnt_alloc

Parameters

x_cnt	2D array of differentials.

See also

x_cnt_alloc

7.5.2.22 void x_cnt_print (uint32_t ∗∗∗ *x_cnt*)

Print the elements of a 2D array of differentials.

Parameters

x_cnt	2D array of differentials.

7.6 include/adp-xor-fi.hh File Reference

Header file for adp-xor-fi.cc: The ADD differential probability of XOR with one fixed input (FI): $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow db)$. .

Defines

- #define ADP_XOR_FI_MSIZE 4
- #define ADP_XOR_FI_NINPUTS 2
- #define ADP_XOR_FI_ISTATE 2
- #define ADP_XOR_FI_NMATRIX 8
- #define ADP_XOR_FI_COLSUM 2
- #define ADP_XOR_FI_NORM 1.0 /(double)ADP_XOR_FI_COLSUM

Functions

- void adp_xor_fixed_input_alloc_matrices (gsl_matrix ∗A[2][2][2])
- void adp_xor_fixed_input_free_matrices (gsl_matrix ∗A[2][2][2])
- void adp_xor_fixed_input_normalize_matrices (gsl_matrix ∗A[2][2][2])
- void adp_xor_fixed_input_sf (gsl_matrix ∗A[2][2][2])
- double adp_xor_fixed_input (gsl_matrix ∗A[2][2][2], uint32_t a, uint32_t db, uint32_t dc)
- double adp_xor_fixed_input_exper (const uint32_t a, const uint32_t db, const uint32_t dc)

7.6.1 Detailed Description

Header file for adp-xor-fi.cc: The ADD differential probability of XOR with one fixed input (FI): $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow db)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.6.2 Define Documentation

7.6.2.1 #define ADP_XOR_FI_COLSUM 2

Sum of non-zero elements in one column of the $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ matrices.

7.6.2.2 #define ADP_XOR_FI_ISTATE 2

Initial state for computing the $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ S-function.

7.6.2.3 #define ADP_XOR_FI_MSIZE 4

Number of state values in the $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ S-function.

7.6.2.4 #define ADP_XOR_FI_NINPUTS 2

Number of inputs to the XOR operation.

7.6.2.5 #define ADP_XOR_FI_NMATRIX 8

Number of $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ matrices.

7.6.2.6 #define ADP_XOR_FI_NORM 1.0 /(double)ADP_XOR_FI_COLSUM

Normalization factor for the $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ matrices.

7.6.3 Function Documentation

7.6.3.1 double adp_xor_fixed_input (gsl_matrix ∗ *A[2][2][2]*, uint32_t *a*, uint32_t *db*, uint32_t *dc*)

The additive differential probability (ADP) of $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$. **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ computed with adp_xor_fixed_input_sf.
a	input value.
db	input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(a, db \to db)$.

7.6.3.2 void adp_xor_fixed_input_alloc_matrices (gsl_matrix ∗ *A[2][2][2]*)

Allocate memory for the transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

See also

adp_xor_fixed_input_free_matrices

7.6.3.3 double adp_xor_fixed_input_exper (const uint32_t *a*, const uint32_t *db*, const uint32_t *dc*)

The additive differential probability (ADP) of $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ computed experimentally over all inputs. Complexity: $O(2^n)$.

Parameters

a	input value.
db	input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(a, db \rightarrow db)$.

See also

adp_xor_fixed_input

7.6.3.4 void adp_xor_fixed_input_free_matrices (gsl_matrix ∗ *A[2][2][2]*)

Free memory reserved by a previous call to adp_xor_fixed_input_alloc_matrices.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

7.6.3.5 void adp_xor_fixed_input_normalize_matrices (gsl_matrix ∗ *A[2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

7.6.3.6 void adp_xor_fixed_input_sf (gsl_matrix ∗ *A[2][2][2]*)

S-function for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$: $\mathrm{adp}^{\oplus}(a, db \rightarrow db)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A.

A[2][2][2] = A[a[i]][db[i]][dc[i]], where

- a[i] : the i-th bit of the fixed input.
- db[i] : the i-th bit of the input difference.
- dc[i] : the i-th bit of the output difference.

7.7 include/adp-xor-pddt.hh File Reference

Header file for adp-xor-pddt.cc. Compute a partial difference distribution table (pDDT) for $\mathrm{adp}^{\oplus}$. .

Functions

- uint32_t adp_xor_ddt_exper (std::multiset< differential_3d_t, struct_comp_diff_3d_p > *diff_set, double p_thres)
- void adp_xor_pddt_i (const uint32_t k, const uint32_t n, const double p_thres, gsl_matrix *A[2][2][2], gsl_vector *C, uint32_t *da, uint32_t *db, uint32_t *dc, double *p, std::multiset< differential_3d_t, struct_comp_diff_3d_p > *diff_set)
- void adp_xor_ddt (uint32_t n, double p_thres)

7.7.1 Detailed Description

Header file for adp-xor-pddt.cc. Compute a partial difference distribution table (pDDT) for $\mathrm{adp}^{\oplus}$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.7.2 Function Documentation

7.7.2.1 void adp_xor_ddt (uint32_t *n*, double *p_thres*)

Compute a partial DDT for $\mathrm{adp}^{\oplus}$: wrapper function of adp_xor_pddt_i.

Parameters

n	word size.
p_thres	probability threshold.

See also

adp_xor_pddt_i.

7.7.2.2 uint32_t adp_xor_ddt_exper (std::multiset< differential_3d_t, struct_comp_diff_3d_p > * *diff_set*, double *p_thres*)

Compute a partial DDT for $\mathrm{adp}^{\oplus}$ by exhaustive search over all input and output differences.

Parameters

diff_set	set of all differentials with probability not less than the threshold (the pDDT)
p_thres	probability threshold.

Returns

number of elements in the pDDT.

See also

adp_xor_pddt_i

7.7.2.3 void adp_xor_pddt_i (const uint32_t *k*, const uint32_t *n*, const double *p_thres*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, double ∗ *p*, std::multiset< differential_3d_t, struct_comp_diff_3d_p > ∗ *diff_set*)

Recursively compute all ADD differentials $(da, db \rightarrow dc)$ for XOR that have probability $\mathrm{adp}^{\oplus}$ larger than a fixed probability threshold `p_thres`.

The function works recursively starting from the LS bit `k` = 0 and terminating at the -MS bit `n`. At every bit position i it assigns values to the i-th bits of the differences `da`, `db`, `dc` and evaluates the probability of the resulting partial (i+1)-bit differential: $(da[i:0], db[i:0] \rightarrow dc[i:0])$. The recursion proceeds only if this probability is not less than the threshold `p_thres`. When i = n, the differential $(da[n-1:0], db[n-1:0] \rightarrow dc[n-1:0])$ is stored in an STL multiset structure (internally implemented as a Red-Black tree).

The **complexity** is strongly dependent on the threshold and is worst-case exponential in the word size: $O(2^{3n})$.

Note

If `p_thres` = 0.0 then the full DDT is computed.
Can be used also to compute all differentials that have non-zero probability by setting p_thres > 0.0 .
For 32 bit words, recommended values for the threshold are p_thres >= 0.5.

Parameters

k	current bit position in the recursion.
n	word size.
p_thres	probability threshold.
A	transition probability matrices for $\mathrm{adp}^{\oplus}$.
C	unit column vector for computing $\mathrm{adp}^{\oplus}$ (adp_xor).
da	first input difference.
db	second input difference.
dc	output difference.
p	probability of the differential $(da[k:0], db[k:0] \rightarrow dc[k:0])$.
diff_set	set of all differentials with probability not less than the threshold (the pDDT)

7.8 include/adp-xor.hh File Reference

Header file for adp-xor.cc: The ADD differential probability of XOR $\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.

.

Defines

- #define ADP_XOR_MSIZE 8
- #define ADP_XOR_NMATRIX 8
- #define ADP_XOR_NINPUTS 2
- #define ADP_XOR_ISTATE 4
- #define ADP_XOR_COLSUM 4
- #define ADP_XOR_NORM 1.0 /(double)ADP_XOR_COLSUM

Functions

- void adp_xor_alloc_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_free_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_normalize_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_print_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_sf (gsl_matrix *A[2][2][2])
- double adp_xor (gsl_matrix *A[2][2][2], uint32_t da, uint32_t db, uint32_t dc)
- double adp_xor_exper (const uint32_t da, const uint32_t db, const uint32_t dc)

7.8.1 Detailed Description

Header file for adp-xor.cc: The ADD differential probability of XOR $\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.

.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.8.2 Define Documentation

7.8.2.1 #define ADP_XOR_COLSUM 4

Sum of non-zero elements in one column of the $\mathrm{adp}^{\oplus}$ matrices.

7.8.2.2 #define ADP_XOR_ISTATE 4

Initial state for computing the $\mathrm{adp}^{\oplus}$ S-function.

7.8.2.3 #define ADP_XOR_MSIZE 8

Number of state values in the $\mathrm{adp}^{\oplus}$ S-function.

7.8.2.4 #define ADP_XOR_NINPUTS 2

Number of inputs to the XOR operation.

7.8.2.5 #define ADP_XOR_NMATRIX 8

Number of $\mathrm{adp}^{\oplus}$ matrices.

7.8.2.6 #define ADP_XOR_NORM 1.0 /(double)ADP_XOR_COLSUM

Normalization factor for the $\mathrm{adp}^{\oplus}$ matrices.

7.8.3 Function Documentation

7.8.3.1 double adp_xor (gsl_matrix ∗ *A[2][2][2]*, uint32_t *da*, uint32_t *db*, uint32_t *dc*)

The additive differential probability of XOR ($\mathrm{adp}^{\oplus}$). **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$ computed with adp_xor_sf.
da	first input difference.
db	second input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.

See also

xdp_add

7.8.3.2 void adp_xor_alloc_matrices (gsl_matrix ∗ *A[2][2][2]*)

Allocate memory for the transition probability matrices for $\mathrm{adp}^{\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

See also

adp_xor_free_matrices

7.8.3.3 double adp_xor_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*)

The additive differential probability of XOR ($\mathrm{adp}^{\oplus}$) computed experimentally over all inputs. **Complexity:** $O(2^{2n})$.

Parameters

da	first input difference.
db	second input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(da, db \to db)$.

See also

adp_xor

7.8.3.4 void adp_xor_free_matrices (gsl_matrix ∗ *A[2][2][2]*)

Free memory reserved by a previous call to adp_xor_alloc_matrices.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

7.8.3.5 void adp_xor_normalize_matrices (gsl_matrix ∗ *A[2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

7.8.3.6 void adp_xor_print_matrices (gsl_matrix ∗ *A[2][2][2]*)

Print the matrices for $\mathrm{adp}^{\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

7.8.3.7 void adp_xor_sf (gsl_matrix ∗ *A[2][2][2]*)

S-function for $\mathrm{adp}^{\oplus}$: $\mathrm{adp}^{\oplus}(da, db \to db)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\mathrm{adp}^{\oplus}(da,db \rightarrow db)$.

$A[2][2][2] = A[da[i]][db[i]][dc[i]]$, where

- $da[i]$: the i-th bit of the first input difference.
- $db[i]$: the i-th bit of the second input difference.
- $dc[i]$: the i-th bit of the output difference.

See also

xdp_add_sf

7.9 include/adp-xor3.hh File Reference

Header file for adp-xor3.cc: The ADD differential probability of XOR with three inputs ($3\oplus$): $\mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$. .

Defines

- #define ADP_XOR3_MSIZE 16
- #define ADP_XOR3_NMATRIX 16
- #define ADP_XOR3_NINPUTS 3
- #define ADP_XOR3_ISTATE 8
- #define ADP_XOR3_COLSUM 8
- #define ADP_XOR3_NORM 1.0 /(double)ADP_XOR3_COLSUM

Functions

- void adp_xor3_alloc_matrices (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_free_matrices (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_print_matrices (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_print_matrices_sage (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_normalize_matrices (gsl_matrix ∗A[2][2][2][2])
- int adp_xor3_states_to_index (int s1, int s2, int s3, int s4)
- void adp_xor3_sf (gsl_matrix ∗A[2][2][2][2])
- double adp_xor3 (gsl_matrix ∗A[2][2][2][2], uint32_t da, uint32_t db, uint32_t dc, uint32_t dd)
- double adp_xor3_exper (const uint32_t da, const uint32_t db, const uint32_t dc, const uint32_t dd)

7.9.1 Detailed Description

Header file for adp-xor3.cc: The ADD differential probability of XOR with three inputs ($3\oplus$): $\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.9.2 Define Documentation

7.9.2.1 #define ADP_XOR3_COLSUM 8

Sum of non-zero elements in one column of the $3\oplus$ matrices.

7.9.2.2 #define ADP_XOR3_ISTATE 8

Initial state for computing the $\mathrm{adp}^{3\oplus}$ S-function.

7.9.2.3 #define ADP_XOR3_MSIZE 16

Number of state values in the $\mathrm{adp}^{3\oplus}$ S-functions.

7.9.2.4 #define ADP_XOR3_NINPUTS 3

Number of inputs to the $3\oplus$ operation.

7.9.2.5 #define ADP_XOR3_NMATRIX 16

Number of $\mathrm{adp}^{3\oplus}$ matrices.

7.9.2.6 #define ADP_XOR3_NORM 1.0 /(double)ADP_XOR3_COLSUM

Normalization factor for the $\mathrm{adp}^{3\oplus}$ matrices.

7.9.3 Function Documentation

7.9.3.1 double adp_xor3 (gsl_matrix ∗ *A[2][2][2][2]*, uint32_t *da*, uint32_t *db*, uint32_t *dc*, uint32_t *dd*)

The additive differential probability (ADP) of $\mathrm{adp}^{3\oplus}$. **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ computed with adp_xor3_sf.
da	first input difference.

db	second input difference.
dc	third input difference.
dd	output difference.

Returns

$\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

See also

adp_xor

7.9.3.2 void **adp_xor3_alloc_matrices** (gsl_matrix ∗ *A[2][2][2][2]*)

Allocate memory for the transition probability matrices for $\mathrm{adp}^{3\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

See also

adp_xor3_free_matrices

7.9.3.3 double **adp_xor3_exper** (const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, const uint32_t *dd*)

The additive differential probability (ADP) of $\mathrm{adp}^{3\oplus}$ computed experimentally over all inputs. **Complexity:** $O(2^{3n})$.

Parameters

da	first input difference.
db	second input difference.
dc	third input difference.
dd	output difference.

Returns

$\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

See also

adp_xor

7.9.3.4 void adp_xor3_free_matrices (gsl_matrix * *A[2][2][2][2]*)

Free memory reserved by a previous call to adp_xor3_alloc_matrices.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.9.3.5 void adp_xor3_normalize_matrices (gsl_matrix * *A[2][2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.9.3.6 void adp_xor3_print_matrices (gsl_matrix * *A[2][2][2][2]*)

Print the matrices for $\mathrm{adp}^{3\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.9.3.7 void adp_xor3_print_matrices_sage (gsl_matrix * *A[2][2][2][2]*)

Print the matrices for $\mathrm{adp}^{3\oplus}$ in a format readable by the computer algebra system Sage (`http://www.sagemath.org/`).

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.9.3.8 void adp_xor3_sf (gsl_matrix * *A[2][2][2][2]*)

S-function for $\mathrm{adp}^{3\oplus}$: $\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

$A[2][2][2][2] = A[da[i]][db[i]][dc[i]][dd[i]]$, where

- $da[i]$: the i-th bit of the first input difference.
- $db[i]$: the i-th bit of the second input difference.

- $dc[i]$: the i-th bit of the third input difference.
- $dd[i]$: the i-th bit of the output difference.

See also

adp_xor_sf

7.9.3.9 int adp_xor3_states_to_index (int *s1*, int *s2*, int *s3*, int *s4*)

Transform the values of the four states of the S-function for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf) into an index.

Parameters

s1	state corresponding to the first input difference.
s2	state corresponding to the second input difference.
s3	state corresponding to the third input difference.
s4	state corresponding to the output difference.

Returns

the index $i = (s_4 + 1)2^3 + s_3 2^2 + s_2 2 + s_1$

7.10 include/adp-xtea-f-fk.hh File Reference

Header file for adp-xtea-f-fk.cc: The ADD differential probability of the F-function of XT-EA for a fixed key and round constants $\mathrm{adp}^F(k, \delta | \; da \rightarrow dd)$. Complexity: $O(n) < c \leq O(2^n)$. .

Functions

- double adp_xtea_f_exper (const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double adp_xtea_f_approx (const uint32_t ninputs, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double max_dy_adp_xtea_f_exper (const uint32_t dx, uint32_t ∗dy_max, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_-const)
- double max_dx_adp_xtea_f_exper (uint32_t ∗dx_max, const uint32_t dy, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_-const)
- double adp_xtea_f_lxr_exper (const uint32_t da, const uint32_t db, uint32_t lsh-_const, uint32_t rsh_const)
- double adp_xtea_f_lxr_approx (const uint32_t ninputs, const uint32_t da, const uint32_t db, uint32_t lsh_const, uint32_t rsh_const)

- bool adp_xtea_f_lxr_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool adp_xtea_f_lxr_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, int32_t x)
- uint32_t adp_xtea_f_lxr_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_-const, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double adp_xtea_f_lxr (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t lsh_const, const uint32_t rsh_const)
- double adp_xtea_f_approx (const uint32_t n, gsl_matrix ∗A[2][2][2], const uint32-_t dx, const uint32_t dy, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- bool adp_xtea_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool adp_xtea_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- uint32_t adp_xtea_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- uint32_t adp_xtea_f_assign_bit_x_dx (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_-const, const uint32_t key, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dx)
- uint32_t adp_xtea_f_assign_bit_x_dy (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_-const, const uint32_t key, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dy)
- double adp_xtea_f (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_-const)
- double max_dy_adp_xtea_f (const uint32_t n, const uint32_t dx, uint32_t ∗ret_dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double max_dx_adp_xtea_f (const uint32_t n, uint32_t ∗ret_dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double first_nz_adp_xtea_f (gsl_matrix ∗A[2][2][2], gsl_matrix ∗AA[2][2][2], const uint32_t key, const uint32_t delta, const uint32_t da, uint32_t ∗ret_dd, uint32_t lsh_const, uint32_t rsh_const)

7.10.1 Detailed Description

Header file for adp-xtea-f-fk.cc: The ADD differential probability of the F-function of XT-EA for a fixed key and round constants $\mathrm{adp}^F(k, \delta |\ da \rightarrow dd)$. Complexity: $O(n) < c \leq O(2^n)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.10.2 Function Documentation

7.10.2.1 double adp_xtea_f (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher XTEA: $\mathrm{adp}^F(k, \delta | \, dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k, \delta | \, dx \rightarrow dy)$.

See also

adp_f_assign_bit_x

7.10.2.2 double adp_xtea_f_approx (const uint32_t *ninputs*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the ADP of the XTEA F-function (xtea_f) obtained over a number of input chosen plaintext pairs chosen uniformly at random.

Parameters

ninputs	number of chosen plaintext pairs.
da	input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k,\delta|\ dx \rightarrow dy)$.

Note

For the exact computation refer to adp_xtea_f

7.10.2.3 double **adp_xtea_f_approx** (const uint32_t *n*, gsl_matrix ∗ *A[2][2][2]*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the ADD differential probability (ADP) of the XTEA F-function (xtea_f) with fixed round key and round cnstant, obtained as the multiplication the ADP of its f_{LXR} component (adp_xtea_f_lxr) and the ADP of XOR with one fixed input (adp_xor_fixed_input):

$\mathrm{adp}^F(k,\delta|\ dx \rightarrow dy) = \mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dt) \cdot \mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dx+dt \rightarrow dy)$.

Algorithm sketch:

1. Compute dz s.t. $p_1 = \max_{dz} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dy \rightarrow dz)$.

   Note

   Note that $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dy \rightarrow dz) = \mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dz \rightarrow dy)$.

2. Compute the output from $f_{\mathrm{LXR}} : dt = dz - dx$.
3. Compute $p_2 = \mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dt)$.
4. Compute $\mathrm{adp}^F(k,\delta|\ dx \rightarrow dy) = p_1 \cdot p_2$.

Parameters

n	word size.
A	transition probability matrices for $\mathrm{adp}^{\oplus}$ with FI (adp_xor_fixed_input_sf).
dx	input difference.
dy	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}} \cdot \mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

Note

For the exact computation refer to adp_xtea_f.

7.10.2.4 uint32_t adp_xtea_f_assign_bit_x (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the F-function of XTEA is satisfied. The function operates by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_xtea_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key ADD probability of `F`: $\mathrm{adp}^F(k, \delta| \; dx \rightarrow dy)$.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_fk

7.10.2.5 uint32_t adp_xtea_f_assign_bit_x_dx (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dx*)

For given output difference `dy`, compute all input differences `dx` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability input difference.

The function works by recursively assigning the bits of `x` and `dx` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat .

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i + rsh_const`).
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dx`.
ret_prob	the maximum probability over all input differences $\max_{dx} \mathrm{adp}^F(k, \delta | \, dx \rightarrow dy)$.
ret_dx	the input difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_assign_bit_x

7.10.2.6 uint32_t **adp_xtea_f_assign_bit_x_dy** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dy*)

For given input difference `dx`, compute all output differences `dy` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability output difference.

The function works by recursively assigning the bits of `x` and `dy` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i + rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
key	round key.
delta	round constant.
dx	input difference.

dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dy`.
ret_prob	the maximum probability over all output differences $\max_{dy} \text{adp}^F(k, \delta \mid dx \rightarrow dy)$.
ret_dy	the output difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_assign_bit_x_dx

7.10.2.7 bool adp_xtea_f_check_x (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value `x` satisfies the ADD differential $(dx \rightarrow dy)$ for the XTEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k, \delta : \; dy = F(x+dx) - F(x)$.

7.10.2.8 double adp_xtea_f_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher XTEA: $\text{adp}^F(k, \delta \mid dx \rightarrow dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k	round key.

delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k,\delta|\, dx \rightarrow dy)$.

7.10.2.9 bool adp_xtea_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` (xtea_f) is satisfied on the `i` LS bits of `x` i.e. check if

$k,\delta:\ dy[i-1:0] = F(x[i-1:0]+dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i + rsh_const`).

Returns

TRUE if $k,\delta:\ dy[i-1:0] = F(x[i-1:0]+dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

7.10.2.10 double adp_xtea_f_lxr (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the ADD differential probability of the f_{LXR} (xtea_f_lxr) function-: $\mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dy)$. **Complexity** c: $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dy)$.

See also

adp_xtea_f_lxr_assign_bit_x

7.10.2.11 double adp_xtea_f_lxr_approx (const uint32_t *ninputs*, const uint32_t *da*, const uint32_t *db*, uint32_t *lsh_const*, uint32_t *rsh_const*)

An approximation of the ADP of f_{LXR} (xtea_f_lxr) obtained over a number of input chosen plaintext pairs chosen uniformly at random.

Parameters

ninputs	number of input chosen plaintext pairs.
da	input difference.
db	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}}(da \rightarrow db)$

Note

For the exact computation refer to adp_xtea_f_lxr_exper

7.10.2.12 uint32_t adp_xtea_f_lxr_assign_bit_x (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, uint32_t ∗ *x_cnt*, double ∗ *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the f_{LXR} (xtea_f_lxr) function is satisfied. The algorithm works by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_xtea_f_lxr_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.

dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the probability $\mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dy)$.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_xtea_f_lxr

7.10.2.13 bool adp_xtea_f_lxr_check_x (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value x satisfies the ADD differential $(dx \rightarrow dy)$ for the function f_{LXR} (xtea_f_lxr).

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $dy = f_{\mathrm{LXR}}(x+dx) - f_{\mathrm{LXR}}(x)$.

7.10.2.14 double adp_xtea_f_lxr_exper (const uint32_t *da*, const uint32_t *db*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the ADD differential probability of the f_{LXR} (xtea_f_lxr) component of the - F-function of block cipher XTEA, through exhaustive search over all input values. - **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}}(da \rightarrow db)$

7.10.2.15 bool adp_xtea_f_lxr_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, int32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for the function f_{LXR} (xtea_f_lxr) is satisfied on the `i` LS bits of `x` i.e. check if

$dy[i-1:0] = f_{\mathrm{LXR}}(x[i-1:0] + dx[i-1:0]) - f_{\mathrm{LXR}}(x[i-1:0]) \bmod 2^i$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of f_{LXR}.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i` + `rsh_const`).

Returns

TRUE if $dy[i-1:0] = f_{\mathrm{LXR}}(x[i-1:0] + dx[i-1:0]) - f_{\mathrm{LXR}}(x[i-1:0]) \bmod 2^i$.

7.10.2.16 double first_nz_adp_xtea_f (gsl_matrix ∗ *A[2][2][2]*, gsl_matrix ∗ *AA[2][2][2]*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *da*, uint32_t ∗ *ret_dd*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For the XTEA F-function (xtea_f), for fixed input difference `da`, compute an arbitrary `dd` such that the differential $(da \rightarrow dd)$ has non-zero probability.

The procedure approximates the ADP of the TEA F-function as a multiplication of the A-DP of its three non-linear components (w.r.t. ADD differences): the two XOR operations and the RSH operation (see xtea_f):

$\mathrm{adp}^F(k,\delta|\, dx \rightarrow dy) = \mathrm{adp}^{\oplus} \cdot \mathrm{adp}^{\gg} \cdot \mathrm{adp}^{\oplus}_{\mathrm{FI}}$

Algorithm sketch:

1. Compute $dy : \max_{dc[i]} \mathrm{adp}^{\oplus}(db, dc[i] \rightarrow dy)$, where $dc[i] \in \{(da \gg 5), (da \gg 5)+1, (da \gg 5) - 2^{n-5}, (da \gg 5) - 2^{n-5}+1\}$, is one of the four possible ADD differences after RSH (adp_rsh).
2. Compute $dt = dy + da$.
3. Compute $dd : \max_{dd} \mathrm{adp}^{\oplus}((k+\delta), dt \rightarrow dd)$.

4. For the computed da and dd experimenttaly re-adjust the probability using adp-_xtea_f_approx.

   Note

   At this step the *exact* probability can also be computed with adp_xtea_f which is more accurate but less efficient.

5. Return the adjusted probability p and dd.

Attention

it is still possible that p = 0.0 for some da.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for $\mathrm{adp}^{\oplus}$ with FI (adp_xor_fixed_input_-sf).
key	round key.
delta	round constant.
da	input difference.
ret_dd	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k, \delta | \; da \rightarrow dd)$

7.10.2.17 double max_dx_adp_xtea_f (const uint32_t *n*, uint32_t * *ret_dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given output difference dy, compute the maximum probability input differences dx over all input differences: $\max_{dx} \mathrm{adp}^F(k, \delta | \; dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

n	word size.
ret_dx	maximum probability input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k, \delta |\ dx \rightarrow dy)$.

See also

max_dx_adp_f_fk

7.10.2.18 double max_dx_adp_xtea_f_exper (uint32_t * *dx_max,* const uint32_t *dy,* const uint32_t *k,* const uint32_t *delta,* const uint32_t *lsh_const,* const uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{adp}^F(k, \delta |\ dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx_max	maximum probability input difference.
dy	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k, \delta |\ dx \rightarrow dy)$.

See also

max_dx_adp_f_fk

7.10.2.19 double max_dy_adp_xtea_f (const uint32_t *n,* const uint32_t *dx,* uint32_t * *ret_dy,* const uint32_t *key,* const uint32_t *delta,* const uint32_t *lsh_const,* const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{adp}^F(k, \delta |\ dx \rightarrow dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory requirement:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k, \delta | \, dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dy, max_dx_adp_f_fk

7.10.2.20 double max_dy_adp_xtea_f_exper (const uint32_t *dx*, uint32_t * *dy_max*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{adp}^F(k, \delta | \, dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx	input difference.
dy_max	maximum probability output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k, \delta | \, dx \rightarrow dy)$.

See also

max_dy_adp_f_fk

7.11 include/common.hh File Reference

Header file for common.cc. Common functions used accross all YAARX programs. .

`#include <iostream> #include <cassert> #include <math.h> #include <gsl/gsl_blas.h> #include <algorithm> #include <vector> #include <set>`

Data Structures

- struct difference_t
- struct differential_3d_t
- struct differential_t
- struct struct_comp_diff_3d_p
- struct struct_comp_diff_p
- struct struct_comp_diff_dx_dy

Defines

- #define IOSTREAM_H
- #define CASSERT_H
- #define MATH_H
- #define GSL_BLAS_H
- #define STL_ALGORITHM_H
- #define STL_VECTOR_H
- #define STL_SET_H
- #define WORD_SIZE 5
- #define ALL_WORDS (1ULL << WORD_SIZE)
- #define MASK (0xffffffff >> (32 - WORD_SIZE))
- #define MOD (1ULL << WORD_SIZE)
- #define TEA_LSH_CONST 1
- #define TEA_RSH_CONST 2
- #define DELTA_INIT 0x9e3779b9
- #define NPAIRS (1ULL << 15)
- #define NROUNDS 10
- #define NDELTA (NROUNDS / 2)
- #define XOR(x, y) ((x ^ y) & MASK)
- #define ADD(x, y) ((x + y) & MASK)
- #define SUB(x, y) ((uint32_t)(x - y + MOD) & MASK)
- #define LSH(x, r) ((x << r) & MASK)
- #define RSH(x, r) ((x >> r) & MASK)
- #define DEBUG_XDP_ADD_TESTS 0
- #define **DEBUG_MAX_XDP_ADD_TESTS** 0
- #define **DEBUG_ADP_XOR_TESTS** 0
- #define **DEBUG_ADP_XOR3_TESTS** 0
- #define **DEBUG_MAX_ADP_XOR_TESTS** 0
- #define **DEBUG_ADP_XOR_FI_TESTS** 0
- #define **DEBUG_MAX_ADP_XOR_FI_TESTS** 0
- #define **DEBUG_MAX_ADP_XOR3_TESTS** 0
- #define **DEBUG_MAX_ADP_XOR3_SET_TESTS** 0
- #define **DEBUG_ADP_RSH_XOR_TESTS** 0
- #define **DEBUG_ADP_SHIFT_TESTS** 0
- #define **DEBUG_EADP_TEA_F_TESTS** 0
- #define **DEBUG_ADP_TEA_F_FK_TESTS** 0
- #define **DEBUG_XDP_TEA_F_FK_TESTS** 0
- #define **DEBUG_XDP_XTEA_F_FK_TESTS** 0
- #define **DEBUG_ADP_XTEA_F_FK_TESTS** 0
- #define DEBUG_ADP_RSH_XOR 0
- #define **DEBUG_ADP_TEA_F_FK** 0
- #define **DEBUG_XDP_TEA_F_FK** 0

Functions

- uint32_t random32 ()
- uint32_t hw8 (uint32_t x)
- uint32_t hw32 (uint32_t x)
- bool is_even (uint32_t i)
- uint32_t gen_sparse (uint32_t hw, uint32_t n)
- void print_binary (uint32_t n)
- bool operator== (differential_t a, differential_t b)
- bool operator< (differential_t x, differential_t y)
- void print_set (const std::set< differential_t, struct_comp_diff_dx_dy > diff_set_-dx_dy)
- void print_mset (const std::multiset< differential_t, struct_comp_diff_p > diff_-mset_p)

7.11.1 Detailed Description

Header file for common.cc. Common functions used accross all YAARX programs. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.11.2 Define Documentation

7.11.2.1 #define ADD(*x*, *y*) ((x + y) & MASK)

The ADD operation on words of size WORD_SIZE

7.11.2.2 #define ALL_WORDS (1ULL << WORD_SIZE)

Total number of words of size WORD_SIZE.

7.11.2.3 #define CASSERT_H

C++ cassert

7.11.2.4 #define DEBUG_ADP_RSH_XOR 0

DEBUG flags for source files.

7.11.2.5 #define DEBUG_XDP_ADD_TESTS 0

DEBUG flags for test files.

7.11.2.6 #define DELTA_INIT 0x9e3779b9

Initial round constant δ of TEA/XTEA.

7.11.2.7 #define GSL_BLAS_H

GSL gsl/gsl_blas.h

7.11.2.8 #define IOSTREAM_H

C++ iostream

7.11.2.9 #define LSH(*x*, *r*) ((x $<<$ r) & MASK)

Left bit shift by r positions on word x of size WORD_SIZE

7.11.2.10 #define MASK (0xffffffff $>>$ (32 - WORD_SIZE))

A mask for the WORD_SIZE LS bits of a 32-bit word.

7.11.2.11 #define MATH_H

math.h

7.11.2.12 #define MOD (1ULL $<<$ WORD_SIZE)

The value 2^{WORD_SIZE}.

7.11.2.13 #define NDELTA (NROUNDS / 2)

Number round constants in TEA/XTEA.

7.11.2.14 #define NPAIRS (1ULL $<<$ 15)

Number of chosen plaintext pairs used in experimentally verifying differential probabilities.

7.11.2.15 #define NROUNDS 10

Number of rounds in reduced-round versions of block ciphers TEA and XTEA.

7.11.2.16 #define RSH(*x*, *r*) ((x $>>$ r) & MASK)

Right bit shift by r positions on word x of size WORD_SIZE

7.11.2.17 #define STL_ALGORITHM_H

STL algorithm

7.11.2.18 #define STL_SET_H

STL set

7.11.2.19 #define STL_VECTOR_H

STL vector

7.11.2.20 #define SUB(x, y) ((uint32_t)(x - y + MOD) & MASK)

The modular subtraction (SUB) operation on words of size WORD_SIZE

7.11.2.21 #define TEA_LSH_CONST 1

Left shift constant of TEA/XTEA.

7.11.2.22 #define TEA_RSH_CONST 2

Right shift constant of TEA/XTEA.

7.11.2.23 #define WORD_SIZE 5

Word size in bits.

7.11.2.24 #define XOR(x, y) ((x ^ y) & MASK)

The XOR operation on words of size WORD_SIZE

7.11.3 Function Documentation

7.11.3.1 uint32_t gen_sparse (uint32_t *hw*, uint32_t *n*)

Generate a random sparse n-bit difference with Hamming weight hw.

7.11.3.2 uint32_t hw32 (uint32_t *x*)

Hamming weight of a 32-bit word.

7.11.3.3 uint32_t hw8 (uint32_t *x*)

Hamming weight of a byte.

7.11.3.4 bool is_even (uint32_t *i*)

Returns true if the argument is an even number.

7.11.3.5 bool operator< (differential_t *x*, differential_t *y*)

Compare two differentials by probability.

7.11.3.6 bool operator== (differential_t *a*, differential_t *b*)

Evaluate if two differentials are identical. Returns TRUE if they are.

7.11.3.7 void print_binary (uint32_t *n*)

Print a value in binary.

7.11.3.8 void print_mset (const std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p*)

Print the list of 2d differentials stored represented as an STL multiset and ordered by probability.

7.11.3.9 void print_set (const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

Print the list of 2d differentials stored represented as an STL set and ordered by index idx = ((2^n dx) + dy), where n is the word size.

7.11.3.10 uint32_t random32 ( )

Generate a random 32-bit value.

7.12 include/eadp-tea-f.hh File Reference

Header file for eadp-tea-f.cc. The expected additive differential probability (EADP) of the F-function of TEA, averaged over all round keys and constants: $\mathrm{eadp}^F(da \rightarrow dd)$. Complexity: $O(n)$. .

Functions

- double eadp_tea_f (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, double ∗prob_db, uint32_t lsh_const, uint32_t rsh_const)
- double eadp_tea_f_exper (const uint32_t dx, const uint32_t dy, uint32_t lsh_-const, uint32_t rsh_const)
- double max_eadp_tea_f (gsl_matrix ∗A[2][2][2][2], const uint32_t da, uint32_t ∗dd_max, double ∗prob_max, uint32_t lsh_const, uint32_t rsh_const)
- double max_eadp_tea_f_exper (gsl_matrix ∗A[2][2][2][2], const uint32_t da, uint32_t ∗dd_max, double ∗prob_max, uint32_t lsh_const, uint32_t rsh_const)
- void nz_eadp_tea_f_i (const uint32_t k, const uint32_t n, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_-t ∗dd, double ∗p, double ∗p_thres, uint32_t ∗ret_dd, double ∗ret_p, uint32_t ∗cnt, uint32_t max_cnt)
- double nz_eadp_tea_f (gsl_matrix ∗A[2][2][2][2], uint32_t da, uint32_t ∗ret_dd)

7.12.1 Detailed Description

Header file for eadp-tea-f.cc. The expected additive differential probability (EADP) of the F-function of TEA, averaged over all round keys and constants: $\mathrm{eadp}^F(da \rightarrow dd)$. Complexity: $O(n)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.12.2 Function Documentation

7.12.2.1 double eadp_tea_f (gsl_matrix * *A[2][2][2][2]*, const uint32_t *da*, const uint32_t *db*, double * *prob_db*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Computing the expected additive differential probability (EADP) of the F-function of TEA, averaged over all round keys and constants. For fixed input and output differences resp. `da` and `db`, it is defined as:

$\mathrm{eadp}^F(da \rightarrow db) = 2^{-4n}\ \{\#(k_0, k_1, \delta, x) : F(x + da) - F(x) = db\}$.

Complexity: $O(n)$.

Algorithm sketch: eadp^F is computed as the multiplication of ADP-s of the two non-linear (w.r.t. XOR differences) components of F, namely XOR and LSH:

$$\mathrm{eadp}^F(da \rightarrow db) = (\sum_{i=0}^{3}(\mathrm{adp}^{\gg 5}(da, dc_i))) \cdot \mathrm{adp}^{3\oplus}_{\mathrm{SET}}((da \ll 4), da, \{dc_0, dc_1, dc_2, dc_3\} \rightarrow db)$$

where $dc_i \in \{(da \gg 5), (da \gg 5) + 1, (da \gg 5) - 2^{n-5}, (da \gg 5) - 2^{n-5} + 1\}$ are the four possible ADD differences after RSH (see adp_rsh) and $\mathrm{adp}^{3\oplus}_{\mathrm{SET}}$ is the ADP of XOR with three inputs where one of the inputs may satisfy any difference from a given *set* (max_adp_xor3_set).

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	input difference.
db	output difference.
prob_db	the expected DP of F.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{eadp}^F(da \rightarrow db)$.

7.12.2.2 double eadp_tea_f_exper (const uint32_t *dx*, const uint32_t *dy*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Computing the expected additive differential probability (EADP) of the F-function of TEA (see eadp_tea_f), experimentally over all round keys and constants.

Complexity: $O(2^{4n})$.

Parameters

dx	input difference.
dy	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{eadp}^F(da \to db)$.

See also

eadp_tea_f

7.12.2.3 double max_eadp_tea_f (gsl_matrix ∗ *A[2][2][2][2],* const uint32_t *da,* uint32_t ∗ *dd_max,* double ∗ *prob_max,* uint32_t *lsh_const,* uint32_t *rsh_const*)

For fixed input difference `da`, compute an output difference `dd` that has maximum expected additive differential probability (EADP) averaged over all round keys and constants of the F-function of TEA:

$\max_{dd} \mathrm{eadp}^F(da \to dd) = 2^{-4n} \, \{\#(k_0, k_1, \delta, x) : F(x + da) - F(x) = dd\}$.

Complexity: $O(n)$.

Algorithm sketch: eadp^F is computed as the multiplication of ADP-s of the two non-linear (w.r.t. XOR differences) components of F, namely XOR and LSH:

$$\mathrm{eadp}^F(da \to dd) = (\sum_{i=0}^{3}(\mathrm{adp}^{\gg 5}(da, dc_i))) \cdot \max_{dd} \mathrm{adp}^{3\oplus}_{\mathrm{SET}}((da \ll 4), da, \{dc_0, dc_1, dc_2, dc_3\} \to dd)$$

where $dc_i \in \{(da \gg 5), (da \gg 5) + 1, (da \gg 5) - 2^{n-5}, (da \gg 5) - 2^{n-5} + 1\}$ are the four possible ADD differences after RSH (see adp_rsh) and $\max_{dd} \mathrm{adp}^{3\oplus}_{\mathrm{SET}}$ is the maximum ADP over all outpt differences, of XOR with three inputs where one of the inputs may satisfy any difference from a given *set* (max_adp_xor3_set).

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	input difference.
dd_max	maximum probability output difference.
prob_max	maximum expected DP of F over all output differences.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{db} \mathrm{eadp}^F(da \rightarrow dd)$.

7.12.2.4 double max_eadp_tea_f_exper (gsl_matrix ∗ *A[2][2][2][2]*, const uint32_t *da*, uint32_t ∗ *dd_max*, double ∗ *prob_max*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Computing the maximum expected additive differential probability (EADP) of the F-function of TEA (see eadp_tea_f), experimentally over all round keys, round constants and output differences.

Complexity: $O(2^{5n})$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	input difference.
dd_max	output difference.
prob_max	the maximum expected DP of F.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{eadp}^F(da \rightarrow db)$.

See also

max_eadp_tea_f

7.12.2.5 double nz_eadp_tea_f (gsl_matrix ∗ *A[2][2][2][2]*, uint32_t *da*, uint32_t ∗ *ret_dd*)

For fixed input diffference `da` to the TEA F-function, generate an arbitrary output difference `dd` for which the expected DP of F is nonzero i.e. $\mathrm{eadp}^F(da \rightarrow dd) > 0$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	first input difference to XOR3.
ret_dd	output difference that is returned as result.

Returns

$\mathrm{eadp}^F(da \rightarrow dd)$.

Attention

Although the resulting differential $(da \rightarrow dd)$ is guaranteed to have expected probability, averaged over all keys and constants, strictly bigger than zero, its probability may still be zero for some fixed value of the round keys and δ constants.

See also

nz_eadp_tea_f_i

7.12.2.6 void nz_eadp_tea_f_i (const uint32_t *k*, const uint32_t *n*, gsl_matrix * *A[2][2][2][2]*, gsl_vector * *C*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t * *dd*, double * *p*, double * *p_thres*, uint32_t * *ret_dd*, double * *ret_p*, uint32_t * *cnt*, uint32_t *max_cnt*)

For fixed input diffferences `da`, `db` and `dc`, to the XOR operation with three inputs in the TEA F-function, generate an arbitrary output difference `dd` for which the expected DP of F is nonzero i.e. $\mathrm{eadp}^F(da \rightarrow dd) > 0$.

Complexity c: $O(n) \leq c \ll O(2^n)$.

Algorithm sketch:

The function works recursively starting from the LS bit `k` = 0 and terminating at the MS bit `n`. At every bit position i it assigns values to the i-th bit of the output difference `dd` and evaluates the probability of the resulting partial (i+1)-bit differential: $(da[i:0], db[i:0], dc[i:0] \rightarrow dd[i:0])$. The recursion proceeds only if this probability is not less than the threshold `p_thres`. When i = n, the difference $dd[n-1:0]$ is stored as the result and the probability $\mathrm{eadp}^F(da \rightarrow dd)$ is returned.

Note

Note that the threshold `p_thres` is initialized to 0.0, but is dynamically updated during the execution as soon as a higher value is found.

Attention

Although the resulting differential $(da \rightarrow dd)$ is guaranteed to have expected probability, averaged over all keys and constants, strictly bigger than zero, its probability may still be zero for some fixed value of the round keys and δ constants.

Parameters

k	current bit position in the recursion.
n	word size.
A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
C	unit column vector for computing $\mathrm{adp}^{3\oplus}$ (adp_xor3).
da	first input difference to XOR3.
db	second input difference to XOR3.
dc	third input difference to XOR3.
dd	output difference from XOR3 (and F).
p	probability of the differential $(da[k:0], db[k:0], dc[k:0] \rightarrow dd[k:0])$.
p_thres	probability threshold.
ret_dd	output difference that is returned as result.
ret_p	the EDP $\mathrm{eadp}^F(da \rightarrow dd)$.
cnt	number of output differences generated so far.
max_cnt	maximum number of output differences allowed (typically 1).

See also

adp_xor_ddt

7.13 include/max-adp-xor-fi.hh File Reference

Header file for max-adp-xor-fi.cc. The maximum ADD differential probability of XOR with one fixed input: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow dc)$. .

Functions

- double max_adp_xor_fixed_input (gsl_matrix ∗A[2][2][2], const uint32_t a, const uint32_t db, uint32_t ∗dd_max)
- double max_adp_xor_fixed_input_exper (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dc_max)

7.13.1 Detailed Description

Header file for max-adp-xor-fi.cc. The maximum ADD differential probability of XOR with one fixed input: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow dc)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.13.2 Function Documentation

7.13.2.1 double max_adp_xor_fixed_input (gsl_matrix ∗ *A[2][2][2],* const uint32_t *a,* const uint32_t *db,* uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(da, db \rightarrow dc)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
a	input value.
db	input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(da, db \rightarrow dc)$.

See also

max_adp_xor_bounds, max_adp_xor_i

7.13.2.2 double max_adp_xor_fixed_input_exper (gsl_matrix ∗ *A[2][2][2],* const uint32_t *da,* const uint32_t *db,* uint32_t ∗ *dc_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	input value.
db	input difference.
dc_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(da, db \rightarrow dc)$.

See also

max_adp_xor_fixed_input

7.14 include/max-adp-xor.hh File Reference

Header file for max-adp-xor.cc. The maximum ADD differential probability of XOR: $\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$. .

Functions

- void max_adp_xor_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C, const uint32_t da, const uint32_t db, uint32_t ∗dd_max, double ∗p_max, uint32_t A_size)
- void max_adp_xor_bounds (gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], const uint32_t da, const uint32_t db, uint32_t ∗dd_max, uint32_t A_size)
- double max_adp_xor (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dd_max)
- double max_adp_xor_exper (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dc_max)

7.14.1 Detailed Description

Header file for max-adp-xor.cc. The maximum ADD differential probability of XOR: $\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.14.2 Function Documentation

7.14.2.1 double max_adp_xor (gsl_matrix ∗ *A[2][2][2]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$.

See also

max_adp_xor_bounds, max_adp_xor_i

7.14.2.2 void max_adp_xor_bounds (gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*, uint32_t *A_size*)

Compute an array of bounds that can be used in the computation of the maximum differential probability.

Parameters

A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

Algorithm Outline:

- Initialize $B[n][i] \leftarrow 1, \; \forall i: \; 0 \leq i < (A_{\mathrm{size}} - 1)$
- For every bit position k from n-1 down to 0
  - For every state `i` from 0 to (`A_size` - 1)
    - Initialize $B[k][i] \leftarrow p_{\max} = 0$
    - Let C^i_{k-1} be a column unit vector of size `A_size` with 1 at position i
    - Recursively assign values to the bits of the output difference `dc` starting at bit popsition $j = k$ and terminating at bit position `n`.
      - The recursion proceeds to bit postion $j+1$ only if the probability p_j of the partially constructed differential $(da[j:k], db[j:k] \rightarrow dc[j:k])$ multiplied by the bound of the probability until the end $B[j+1]$ is bigger than the best probability found so far i.e. if: $B[j+1]A_jA_{j-1}\ldots A_kC^i_{k-1} > p_{\max}$
      - When $j = n$ update the max.: $p_{\max} \leftarrow p_{n-1} = \mathrm{dp}(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$.
    - At the end of the recursion set $B[k][i] \leftarrow p_{\max}$.

Meaning of the bounds B:

$B[k][i]$ is an *upper bound* on on the maximum probability of the differential $(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$ because clearly for any choice $dc[n-1:k]$ of the $(n-k)$ MS bits of dc, the probability $LA_{n-1}A_{n-2}\ldots A_kC^i_{k-1}$ will never be bigger than $B[k][i]$. Furthermore, let $G[k] = LA_{n-1}A_{n-2}\ldots A_k$ be the multiplication of the corresponding transition probability matrices for the $(n-k)$ MS bits of dc $dc[n-1:k]$ and let $H[k-1] = A_{k-1}A_{k-2}\ldots A_0C^i_{k-1}$ and $H[-1] = C$. Then $\mathrm{dp}(da, db \rightarrow dc) = G[k]H[k-1] \leq B[k]H[k-1]$ for *any* choice of $dc[n-1:k]$. In particular, when $k = 0$ $\mathrm{dp}(da, db \rightarrow dc) = \max_{dc} \mathrm{dp}(da, db \rightarrow dc) = G[0]C = B[0]C$.

See also

max_adp_xor_i

7.14.2.3 double max_adp_xor_exper (gsl_matrix ∗ *A[2][2][2]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dc_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$.

See also

max_adp_xor

7.14.2.4 void max_adp_xor_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double ∗ *p*, uint32_t ∗ *dd*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, gsl_vector ∗ *C*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*, double ∗ *p_max*, uint32_t *A_size*)

Compute an *upper bound* $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$ starting from initial state `i` of the S-function i.e. $\mathrm{dp}(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k]) = LA_{n-1}A_{n-2}\ldots A_k C^i_{k-1}$, given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j] \rightarrow dc[n-1:j])$ for $j = k+1, k+2, \ldots, n-1$, where $L = [1\ 1\ \ldots\ 1]$ is a row vector of size `A_size` and C^i_{k-1} is a unit column vector of size `A_size` with 1 at position i and $C^i_{-1} = C$.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the estimated probability at bit position `k`.
dd	output difference.
A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i:` `A_size` $> i \geq 0$.
C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

Algorithm Outline:

Recursively assign values to the bits of the output difference `dc` starting at bit popsition $j = k$ and terminating at bit position `n`. The recursion proceeds to bit postion $j+1$ only if the probability p_j of the partially constructed differential $(da[j:k], db[j:k] \rightarrow dc[j:k])$ multiplied by the bound of the probability until the end $B[j+1]$ is bigger than the best probability found so far i.e. if: $B[j+1]A_jA_{j-1}\ldots A_kC^i_{k-1} > p_{\max}$. When $j = n$ update the max.: $p_{\max} \leftarrow p_{n-1} = \mathrm{dp}(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$.

See also

max_adp_xor_bounds

7.15 include/max-adp-xor3-set.hh File Reference

Header file for max-adp-xor3-set.cc. The maximum ADD differential probability of -XOR with three inputs, where one of the inputs satisfies a *set* of ADD differences: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$. .

Defines

- #define ADP_XOR3_SET_SIZE 4

Functions

- void max_adp_xor3_set_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C[ADP_XOR3_SET_SIZE], const uint32_t da, const uint32_t db, const uint32_t dc[ADP_XOR3_SET_SIZE], uint32_t ∗dd_max, double ∗p_max)
- double max_adp_xor3_set (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc[ADP_XOR3_SET_SIZE], double p_dc[ADP_XOR3_SET_SIZE], uint32_t ∗dd_max)
- double max_adp_xor3_set_exper (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc[ADP_XOR3_SET_SIZE], double p_dc[ADP_XOR3_SET_SIZE], uint32_t ∗dd_max)

7.15.1 Detailed Description

Header file for max-adp-xor3-set.cc. The maximum ADD differential probability of -XOR with three inputs, where one of the inputs satisfies a *set* of ADD differences: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.15.2 Define Documentation

7.15.2.1 #define ADP_XOR3_SET_SIZE 4

Number of input differences in the set.

7.15.3 Function Documentation

7.15.3.1 double max_adp_xor3_set (gsl_matrix ∗ *A[2][2][2][2],* const uint32_t *da,* const uint32_t *db,* const uint32_t *dc[ADP_XOR3_SET_SIZE],* double *p_dc[ADP_XOR3_SET_SIZE],* uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences for a set of input differenecs: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \dots\} \rightarrow dd)$.

Complexity c: $O(nR) \leq c \leq O(2^{nR})$, where R is the size of the set of input differences dc_r.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	set of input difference.
dd_max	maximum probability output difference.
p_dc	probabilities of the set of differentials corresponding to the set of differences (used for testing and debug only).

Returns

$\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \dots\} \rightarrow dd)$.

Algorithm Outline:

- Compute the bounds for each of the differences in the set *independently* using max_adp_xor_bounds i.e. compute $B_r[k]$ - the bounds ror the R differentials: $\mathrm{dp}(da[n-1:k], db[n-1:k], dc_r[n-1:k] \rightarrow dd[n-1:k])$ corresponding to the r-th input differences dc_r in the set.
- Compute a single array of bounds $B_{\max}$ as the maximum of the bounds $B_r[k]$ at every bit position $0 \leq k \leq n$ for every S-function state $0 \leq i < A_{\mathrm{size}}$: $B_{\max}[k][i] = \max_r B[k][i],\ 0 \leq k \leq n,\ 0 \leq i < A_{\mathrm{size}}$.
- Call max_adp_xor3_set_i with the array of bounds $B_{\max}[k][i]$ to compute the final maximum probability $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}$.

See also

max_adp_xor3_set_i, max_adp_xor_bounds, max_adp_xor

7.15.3.2 double max_adp_xor3_set_exper (gsl_matrix ∗ *A[2][2][2][2],* const uint32_t *da,* const uint32_t *db,* const uint32_t *dc[ADP_XOR3_SET_SIZE],* double *p_dc[ADP_XOR3_SET_SIZE],* uint32_t ∗ *dd_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	set of input difference.
dd_max	maximum probability output difference.
p_dc	probabilities of the set of differentials corresponding to the set of differences; normally set to 1 (used for testing and debug only).

Returns

$\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$.

See also

max_adp_xor3_set

7.15.3.3 void max_adp_xor3_set_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double ∗ *p*, uint32_t ∗ *dd*, gsl_matrix ∗ *A[2][2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, gsl_vector ∗ *C[ADP_XOR3_SET_SIZE]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc[ADP_XOR3_SET_SIZE]*, uint32_t ∗ *dd_max*, double ∗ *p_max*)

Compute an upper bound $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k], \{dc_0[n-1:k], dc_1[n-1:k], \ldots\} \rightarrow dd[n-1:k])$, starting from initial state `i` of the S-function and given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j], \{dc_0[n-1:j], dc_1[n-1:j], \ldots\} \rightarrow dd[n-1:j])$ for $j = k+1, k+2, \ldots, n-1$, where $\{dc_0[n-1:k], dc_1[n-1:k], \ldots\}$ is a finite set of input differences.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the estimated probability at bit position `k`.
dd	output difference.
A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dc	set of input differences.
dd_max	maximum probability output difference.
p_max	the maximum probability.

Algorithm Outline:

The bound for the set of differences is computed as the sum of the bounds of the differentials obtained from each of the elements of the set: $B[k][i] = \sum_r B_r[k][i]$, where $B_r[k][i]$ is an upper bound on the maximum probability of the differential corresponding to the r-th input difference dc_r i.e. $\mathrm{dp}(da[n-1:k], db[n-1:k], dc_r[n-1:k] \rightarrow dd[n-1:k])$ computed as in max_adp_xor_i.

See also

max_adp_xor3_set, max_adp_xor_i

7.16 include/max-adp-xor3.hh File Reference

Header file for max-adp-xor3.cc. The maximum ADD differential probability of XOR with three inputs: $\max_{dd} \mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$. .

Functions

- void max_adp_xor3_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max, double ∗p_max)
- void max_adp_xor3_bounds (gsl_matrix ∗A[2][2][2][2], gsl_vector ∗B[WORD_SIZE+1], const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)
- double max_adp_xor3 (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)
- void max_adp_xor3_rec_i (const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max, double ∗p_max)
- double max_adp_xor3_rec (gsl_matrix ∗A[2][2][2][2], gsl_vector ∗C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)
- double max_adp_xor3_exper (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)

7.16.1 Detailed Description

Header file for max-adp-xor3.cc. The maximum ADD differential probability of XOR with three inputs: $\max_{dd} \mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.16.2 Function Documentation

7.16.2.1 double max_adp_xor3 (gsl_matrix * *A[2][2][2][2]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t * *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{adp}^{\oplus}(da, db, dc \rightarrow dd)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

See also

max_adp_xor3_bounds, max_adp_xor3_i

7.16.2.2 void max_adp_xor3_bounds (gsl_matrix * *A[2][2][2][2]*, gsl_vector * *B[WORD_SIZE+1]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t * *dd_max*)

Compute an array of bounds that can be used in the computation of the maximum differential probability.

Parameters

A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

See also

max_adp_xor_bounds, max_adp_xor3_i

7.16.2.3 double max_adp_xor3_exper (gsl_matrix * *A[2][2][2][2]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t * *dd_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dd} \mathrm{adp}^{\oplus}(da, db, dc \rightarrow dd)$

See also

max_adp_xor

7.16.2.4 void max_adp_xor3_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double ∗ *p*, uint32_t ∗ *dd*, gsl_matrix ∗ *A[2][2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, gsl_vector ∗ *C*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t ∗ *dd_max*, double ∗ *p_max*)

Compute an upper bound $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k], dc[n-1:k] \rightarrow dd[n-1:k])$ starting from initial state `i` of the S-function given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j], dc[n-1:j] \rightarrow dd[n-1:j])$ for $j = k+1, k+2, \ldots, n-1$.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the transition probability of state `i` at bit position `k`.
dd	output difference.
A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.

See also

max_adp_xor_i

7.16.2.5 double max_adp_xor3_rec (gsl_matrix ∗ *A[2][2][2][2]*, gsl_vector ∗ *C*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t ∗ *dd_max*)

Recursively compute the maximum differential probability over all output differences: $\max_{dd} \mathrm{adp}^{\oplus}(da,db,dc \rightarrow dd)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
C	unit row vector initialized with 1 at the nitial state.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dd} \mathrm{adp}^{\oplus}(da,db,dc \rightarrow dd)$

Note

This function max_adp_xor3_rec is more efficient than exhaustive search over all output differences max_adp_xor3_exper, but is less efficient than the function max-_adp_xor3 that uses bounds. The reason is that at every bit position, max_adp_-xor3_rec (by max_adp_xor3_rec_i) implicitly assumes that the remaining probability until the end (i.e. until the MSB) is 1, while the bounds computed by max_-adp_xor3 are tighter and thus more branches of the recursion are cut earlier in the computation.

See also: max_adp_xor3_i()

7.16.2.6 void max_adp_xor3_rec_i (const uint32_t *k*, const uint32_t *n*, double ∗ *p*, uint32_t ∗ *dd*, gsl_matrix ∗ *A[2][2][2][2]*, gsl_vector ∗ *C*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t ∗ *dd_max*, double ∗ *p_max*)

Recursively compute the maximum differential probability over all output differences of the partial $(n-k)$-bit differential $\max_{dd} \mathrm{adp}^{\oplus}(da[n-1:k],db[n-1:k],dc[n-1:k] \rightarrow dd[n-1:k])$.

Parameters

k	current bit position: $n > k \geq 0$.
n	word size.
p	the probability at bit position k.
dd	output difference.
A	transition probability matrices.
C	unit row vector initialized with 1 at the nitial state.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.

Algorithm Outline:

The function recursively assigns the bits of the output difference starting at the LS bit position $k = 0$ and proceeding to $k+1$ only if the probability so far is still above the maximum that was found up to now. The initial value for the maximum probability `p_max` is 0 and is updated dynamically during the process every time a higher probability is encountered. The recursion stops at the MSB $k = n$.

See also: max_adp_xor3_rec()

7.17 include/max-xdp-add.hh File Reference

Header file for max-xdp-add.cc. The maximum XOR differential probability of ADD: $\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$. .

Functions

- void max_xdp_add_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C, const uint32_t da, const uint32_t db, uint32_t ∗dd_max, double ∗p_max, uint32_t A_size)
- void max_xdp_add_bounds (gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], const uint32_t da, const uint32_t db, uint32_t ∗dd_max, uint32_t A_size)
- double max_xdp_add (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dd_max)
- double max_xdp_add_exper (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dc_max)

7.17.1 Detailed Description

Header file for max-xdp-add.cc. The maximum XOR differential probability of ADD: $\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.17.2 Function Documentation

7.17.2.1 double max_xdp_add (gsl_matrix ∗ *A[2][2][2],* const uint32_t *da,* const uint32_t *db,* uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$.

See also

max_xdp_add, max_xdp_add_i, max_adp_xor

7.17.2.2 void max_xdp_add_bounds (gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*, uint32_t *A_size*)

Compute an array of bounds that can be used in the computation of the maximum differential probability.

Parameters

A	transition probability matrices.
B	array of size `A_size` rows by $(\mathrm{n}+1)$ columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i` : `A_size` $> i \geq 0$.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

See also

max_xdp_add_i, max_adp_xor_bounds

7.17.2.3 double max_xdp_add_exper (gsl_matrix ∗ *A[2][2][2]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dc_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc_max	maximum probability output difference.

Returns

$\max_{dc} \text{xdp}^{+}(da, db \to dc)$.

See also

max_xdp_add

7.17.2.4 void max_xdp_add_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double ∗ *p*, uint32_t ∗ *dd*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, gsl_vector ∗ *C*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*, double ∗ *p_max*, uint32_t *A_size*)

Compute an *upper bound* $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k] \to dc[n-1:k])$ starting from initial state `i` of the S-function i.e. $\text{dp}(da[n-1:k], db[n-1:k] \to dc[n-1:k]) = LA_{n-1}A_{n-2}\dots A_kC^i_{k-1}$, given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j] \to dc[n-1:j])$ for $j = k+1, k+2, \dots, n-1$, where $L = [1\ 1\ \dots\ 1]$ is a row vector of size `A_size` and C^i_{k-1} is a unit column vector of size `A_size` with 1 at position i and $C^i_{-1} = C$.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the estimated probability at bit position `k`.
dd	output difference.
A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

See also

max_adp_xor_i

7.18 include/tea-add-ddt-search.hh File Reference

Declarations for tea-add-ddt-search.cc. Automatic search for ADD differential trails in TEA using full DDT-s. .

Functions

- double verify_trail (uint64_t npairs, differential_t trail[NROUNDS], uint32_t nrounds, uint32_t key[4], uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- void round_ddt (const int n, const int nrounds, differential_t **RSDDT_E, differential_t **RSDDT_O, differential_t *SDDT_O, double B[NROUNDS], double *Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS])
- void tea_search_ddt (uint32_t key[4])
- void round_xddt (const int n, const int nrounds, differential_t ***XRSDDT_E, differential_t ***XRSDDT_O, differential_t **XSDDT_O, const double B[NROUNDS], double *Bn, differential_t diff[NROUNDS], differential_t trail[NROUNDS])
- void tea_search_xddt (uint32_t key[4])
- void round_xddt_bottom_up (const int n, const int nrounds, differential_t ***XRSDDT_E, differential_t ***XRSDDT_O, differential_t **XSDDT_E, differential_t **XSDDT_O, const double B[NROUNDS], double *Bn, differential_t diff[NROUNDS], differential_t trail[NROUNDS])
- void tea_search_xddt_bottom_up (uint32_t key[4])

7.18.1 Detailed Description

Declarations for tea-add-ddt-search.cc. Automatic search for ADD differential trails in TEA using full DDT-s. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.18.2 Function Documentation

7.18.2.1 void round_ddt (const int *n*, const int *nrounds*, differential_t ** *RSDDT_E*, differential_t ** *RSDDT_O*, differential_t * *SDDT_O*, double *B[NROUNDS]*, double * *Bn*, const differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*)

Automatic search for ADD differential trails using precomputed full difference distribution tables (DDT) for **a modified version of TEA** that uses the same round constant δ in every round.

Attention

1. Assumes the same δ constant is used at every round of TEA.
2. Two DDT-s are computed: `DDT_E` contains fixed-key probabilities for the round keys applied in all even rounds: $0,2,4,\ldots$; `DDT_O` contains fixed-key probabilities for the round keys applied in all odd rounds: $1,3,5,\ldots$.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
RSDDT_E	a DDT for the keys of all even rounds $0,2,4,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (a Row-Sorted `DDT_E`).
RSDDT_O	a DDT for the keys of all odd rounds $1,3,5,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (a Row-Sorted `DDT_O`).
SDDT_E	a DDT for the keys of all even rounds will all elements sorted in descending order of their probability (a Sorted `DDT_E`).
nrounds	total number of rounds (NROUNDS).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best differential trail for `nrounds`.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

See also

tea_add_threshold_search

7.18.2.2 void round_xddt (const int *n*, const int *nrounds*, differential_t $***$ *XRSDDT_E*, differential_t $***$ *XRSDDT_O*, differential_t $**$ *XSDDT_O*, const double *B[NROUNDS]*, double $*$ *Bn*, differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*)

Automatic search for ADD differential trails using precomputed full difference distribution tables (DDT) for **the original version of TEA**.

Attention

For every round constant δ, two DDT-s are computed: `DDT_E` containing the fixed-key fixed- δ probabilities for the round keys applied in all even rounds: $0,2,4,\ldots$ and `DDT_O` containing the fixed-key fixed- δ probabilities for the round keys applied in all odd rounds: $1,3,5,\ldots$. Since δ is updated every second round, for N rounds $2(N/2)$ DDT-s will be computed.

Parameters

n	index of the current round: $0 \leq n < \text{nrounds}$.
nrounds	total number of rounds (NROUNDS).
XRSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds $0, 2, 4, \ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_E`).
XRSDDT_O	an array of fixed-key fixed- δ DDT-s for all odd rounds $1, 3, 5, \ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_O`).
XSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds will all elements sorted in descending order of their probability (an eXtended Sorted `DDT_E`).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best differential trail for `nrounds`.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

See also

round_ddt

7.18.2.3 void round_xddt_bottom_up (const int *n*, const int *nrounds*, differential_t *** *XRSDDT_E*, differential_t *** *XRSDDT_O*, differential_t ** *XSDDT_E*, differential_t ** *XSDDT_O*, const double *B[NROUNDS]*, double * *Bn*, differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*)

Automatic search for ADD differential trails using precomputed full difference distribution tables (DDT) for **the original version of TEA**.

round_xddt_bottom_up is conceptually the same as round_xddt, except that **the search proceeds from the bottom up** i.e. first finds the best 1-round trail for the last round N, next finds the best 2-round trail for rounds $N-1, N$, etc. finds the best i-round trail for rounds $i, i+1, \ldots, N$ and finally finds the best N-round trail.

Attention

For every round constant δ, two DDT-s are computed: `DDT_E` containing the fixed-key fixed- δ probabilities for the round keys applied in all even rounds: $0,2,4,\ldots$ and `DDT_O` containing the fixed-key fixed- δ probabilities for the round keys applied in all odd rounds: $1,3,5,\ldots$. Since δ is updated every second round, for N rounds $2(N/2)$ DDT-s will be computed.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
XRSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds $0,2,4,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_E`).
XRSDDT_O	an array of fixed-key fixed- δ DDT-s for all odd rounds $1,3,5,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_O`).
XSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds will all elements sorted in descending order of their probability (an eXtended Sorted `DDT_E`).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best differential trail for `nrounds`.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for n rounds.
- $B[1]$: best probability for $(n-1)$ rounds.
- $\ldots$
- $B[i]$: best probability for $(n-i)$ rounds (rounds $n-i, n-i+1, \ldots, n$).
- $\ldots$
- $B[n-2]$: best probability for 2 rounds (rounds $n-1, n$).
- $B[n-1]$: best probability for 1 round (round n).

See also

round_xddt

7.18.2.4 void tea_search_ddt (uint32_t *key[4]*)

Search for ADD differential trails in a modified version of block cipher TEA that uses the same round constant δ in every round. Computes full difference distribution tables (DDT) for every key and the same round constant: a wrapper function for round_ddt.

Parameters

key	cryptographic key of TEA.

Attention

Assumes the same δ constant is used at every round of TEA.

See also

tea_add_trail_search

7.18.2.5 void tea_search_xddt (uint32_t *key[4]*)

Search for ADD differential trails in the original version of block cipher TEA. Computes full difference distribution tables (DDT) for every key and every round constant: a wrapper function for round_xddt.

Parameters

key	cryptographic key of TEA.

See also

round_xddt

7.18.2.6 void tea_search_xddt_bottom_up (uint32_t *key[4]*)

Search for ADD differential trails in the original version of block cipher TEA. Computes full difference distribution tables (DDT) for every key and every round constant. - Conceptually the same as tea_search_xddt, except that the search starts from the last round and proceeds up to the first (i.e. in a bottom-up amnner). This function is a wrapper for round_xddt_bottom_up.

Parameters

key	cryptographic key of TEA.

See also

round_xddt_bottom_up

7.18.2.7 double verify_trail (uint64_t *npairs*, differential_t *trail[NROUNDS]*, uint32_t *nrounds*, uint32_t *key[4]*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Experimentally verify the probabilities of the 1-round differentials composing an N-round differential trail for block cipher TEA, against the exact probabilities from a DDT.

Parameters

npairs	number of chosen plaintext pairs (NPAIRS).
trail	best differential trail for `nrounds`.
nrounds	total number of rounds (NROUNDS).
key	cryptographic key of TEA.
delta	round constant.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).

7.19 include/tea-add-threshold-search.hh File Reference

Header for tea-add-threshold-search.cc. Automatic search for ADD differential trails in block cipher TEA. .

Functions

- void tea_add_threshold_search (const int n, const int nrounds, const uint32_t npairs, const uint32_t key[4], gsl_matrix *A[2][2][2][2], double B[NROUNDS], double *Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS], uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void tea_add_trail_search (uint32_t key[4])

7.19.1 Detailed Description

Header for tea-add-threshold-search.cc. Automatic search for ADD differential trails in block cipher TEA. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.19.2 Function Documentation

7.19.2.1 void tea_add_threshold_search (const int *n*, const int *nrounds*, const uint32_t *npairs*, const uint32_t *key[4]*, gsl_matrix $*$ *A[2][2][2][2]*, double *B[NROUNDS]*, double $*$ *Bn*, const **differential_t** *diff_in[NROUNDS]*, **differential_t** *trail[NROUNDS]*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset$<$ **differential_t, struct_comp_diff_p** $>$ $*$ *diff_mset_p*, std::set$<$ **differential_t, struct_comp_diff_dx_dy** $>$ $*$ *diff_set_dx_dy*)

Automatic search for ADD differential trails in block cipher TEA. using pDDT.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
key	cryptographic key of TEA.
A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best found differential trail for `nrounds`.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	set of differentials (dx, dy, p) (the pDDT) ordered by probability p.
diff_set_dx_dy	set of differentials (dx, dy, p) (the pDDT) ordered by index $i = (dx\ 2^n + dy)$.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

Algorithm Outline:

The algorithm is based on Matsui search strategy described in [Sect. 4, Matsui, `On correlation between the order of S-boxes and the strength of DES`, EUROCRYPT'94]. The main idea is to view the F-function of TEA as an S-box for which a partial difference distribution table (pDDT) is constructed (tea_f_add_pddt). Then a recursive search for differential trails over a given number of rounds $n \geq 1$ is performed. From knowledge of the best probabilities $B_1, B_2, \ldots, B_{n-1}$ for the first $(n-1)$ rounds and an initial estimate $\overline{B}_n$ for the probability for n rounds the best probability B_n for n rounds is derived. Note that for the estimate the following must hold: $\overline{B}_n \leq B_n$.

In addition to Matsui's notation for the probability of the best n-round trail B_n and of its estimation $\overline{B}_n$ we introduce $\widehat{B}_n$ to denote the probability of *the best found* trail for n rounds: $\overline{B}_n \leq \widehat{B}_n \leq B_n$. Given a pDDT D of maximum size m, an estimation for the best n-round probability $\overline{B}_n$ with its corresponding n-round differential trail $\overline{T}$ and the probabilities $\widehat{B}_1, \widehat{B}_2, \ldots, \widehat{B}_{n-1}$ of the best found trails for the first $(n-1)$ rounds, tea_-add_threshold_search outputs an n-round trail $\widehat{T}$ that has probability $\widehat{B_n} \geq \overline{B}_n$.

tea_add_threshold_search operates by recursively extending a trail for i rounds to $(i+1)$ rounds, beginning with $i=1$ and terminating at $i=n$. This is done by exploring multiple differential trails constructed from the entries of the pDDT D at every round. If, in the process, a differential that is not already in D is encountered it is added to D, provided that the maximum size m has not been reached. The recursion at level i continues to level $(i+1)$ only if the probability of the constructed i-round trail multiplied by the probability of the best found trail for $(n-i)$ rounds is at least $\overline{B}_n$ i.e. if, $p_1 p_2 \ldots p_i \widehat{B}_{n-i} \geq \overline{B}_n$ holds. For $i=n$ the last equation is equivalent to: $p_1 p_2 \ldots p_n = \widehat{B}_n \geq \overline{B}_n$. If the latter holds, the initial estimate is updated with the new: $\overline{B}_n \leftarrow \widehat{B}_n$ and the corresponding trail is also updated accordingly: $\overline{T}_n \leftarrow \widehat{T}_n$. Upon termination the best found trail $\widehat{T}_n$ and its probability $\widehat{B_n}$ are returned as result.

Termination

The algorithm terminates when one of the following two events happens first:

1. The initial estimate $\overline{B}_n$ can not be improved further.
2. The maximum size m of the pDDT D is reached and all differentials in D in every round have been explored.

Complexity

The complexity of tea_add_threshold_search depends on the following factors:

1. The closeness of the best found probabilities $\widehat{B}_1, \widehat{B}_2, \ldots, \widehat{B}_{n-1}$ for the first $(n-1)$ rounds to the actual best probabilities.
2. The tightness of the initial estimate $\overline{B}_n$.
3. The number of elements in D. The latter is determined by the probability threshold used to compute D and by the maximum number of elements m allowed.

In the worst-case, in every round, except the last, m iterations will be executed. - Therefore the worst-case complexity is $\mathcal{O}(m^{n-1})$, where n is the number of rounds. Although the algorithm is worst-case exponential in the number of rounds, it is much more efficient in practice.

Attention

The algorithm does not guarantee to find the *best* trail.

Note

The pDDT of TEA contains the expected differential probabilities of F averaged over all keys and round constants. To obtain better estimate of the probabilities of trails for a fixed key and round constants, in the process of the search the probability of each differential is additionally adjusted to the value of the round key and constant by performing one-round encryptions over `npairs` pairs of chosen plaintexts.

7.19.2.2 void tea_add_trail_search (uint32_t *key[4]*)

Search for ADD differential trails in block cipher TEA: wrapper function for tea_add_threshold_search.

Parameters

key	cryptographic key of TEA.

Algorithm Outline:

The procedure operates as follows:

1. Compute a pDDT for F (tea_f_add_pddt).
2. Adjust the probabilities of the pDDT to the round key and constant (tea_f_add_pddt_adjust_to_key).
3. Execute the search for differential trails for n rounds (n = NROUNDS) through a successive application of tea_add_threshold_search :
   - Compute the best found probability on 1 round: $B[0]$.
   - Using $B[0]$ compute the best found probability on 2 rounds: $B[1]$.
   - $\ldots$
   - Using $B[0],\ldots,B[i-1]$ compute the best found probability on $(i+1)$ rounds: $B[i]$.
   - $\ldots$
   - Using $B[0],\ldots,B[n-2]$ compute the best found probability on n rounds: $B[n-1]$.
4. Print the best found trail on n rounds on standrad output and terminate.

See also

tea_add_threshold_search

7.20 include/tea-f-add-pddt.hh File Reference

Header file for tea-f-add-pddt.cc. Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA. .

Functions

- bool rsh_condition_is_sat (const uint32_t k, const uint32_t new_da, const uint32_t new_dc)
- bool lsh_condition_is_sat (const uint32_t k, const uint32_t new_da, const uint32_t new_db)
- void tea_f_add_pddt_i (const uint32_t k, const uint32_t n, const uint32_t lsh_const, const uint32_t rsh_const, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗C, uint32_t ∗da, uint32_t ∗db, uint32_t ∗dc, uint32_t ∗dd, double ∗p, const double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy)
- void tea_f_add_pddt (uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy)
- void tea_f_add_pddt_adjust_to_key (uint32_t nrounds, uint32_t npairs, uint32_t key[4], double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy)
- void tea_f_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > ∗diff_mset_p, const std::set< differential_t, struct_comp_diff_dx_dy > diff_set_dx_dy)
- void tea_f_add_pddt_exper (gsl_matrix ∗A[2][2][2][2], uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > ∗diff_mset_p)
- void tea_f_add_pddt_fk_exper (uint32_t n, double p_thres, uint32_t delta, uint32_t k0, uint32_t k1, uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > ∗diff_mset_p)

7.20.1 Detailed Description

Header file for tea-f-add-pddt.cc. Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.20.2 Function Documentation

7.20.2.1 bool lsh_condition_is_sat (const uint32_t *k*, const uint32_t *new_da*, const uint32_t *new_db*)

Check if two differences da and dc, partially constructed up to bit k (WORD_SIZE $> k \geq 0$), are valid input and output difference respectively, for the LSH operation.

Parameters

k	bit position: WORD_SIZE $> k \geq 0$.
new_da	input difference to LSH partially constructed up to bit k.
new_db	output difference from LSH partially constructed up to bit k.

Returns

TRUE if dc, after being fully constructed, will be a valid output difference from LSH, given the input difference da; FALSE otherwise.

More Details:

1. If $k < L$: check if $db[k:0] = 0$.
2. If $k \geq L$: check if $(db \gg L)[n-(k-L+1):0] = da[n-(k-L+1):0]$.

where $L =$TEA_LSH_CONST, $n =$WORD_SIZE.

See also

rsh_condition_is_sat

7.20.2.2 bool rsh_condition_is_sat (const uint32_t *k*, const uint32_t *new_da*, const uint32_t *new_dc*)

Check if two differences da and dc, partially constructed up to bit k (WORD_SIZE $>$ $k \geq 0$), are valid input and output difference respectively, for the RSH operation. From the partial information for dc, the algorithm estimates if dc belongs to one of the four possible differences after the RSH operation (see adp_rsh): $\{(da \gg R),\ (da \gg R)+1,\ (da \gg R)-2^{n-R},\ (da \gg R)-2^{n-R}+1\}$, where R is the RSH constant (TEA_RSH-_CONST).

Parameters

k	bit position: WORD_SIZE $> k \geq 0$.
new_da	input difference to RSH partially constructed up to bit k.
new_dc	output difference from RSH partially constructed up to bit k.

Returns

TRUE if dc, after being fully constructed, will be a valid output difference from RSH, given the input difference da; FALSE otherwise.

Attention

The function is *not* optimal, meaning that it is overly-restrictive: all diferences (da, dc) which pass the checks are valid, but there also exist valid differences that do not pass the checks. The reason is that it is hard to detect all valid differences before they have been fully constructed.

More Details:

Given are two differences da and dc, that are only partially constructed up to bit k (counting from the LSB $k = 0$). rsh_condition_is_sat performs checks on da and dc and outputs if dc is such that $dc = da \gg R$, where R = TEA_RSH_CONST. The idea is to be able to discard pairs of diferences (da, dc) before they have been fully constructed. This allows to more efficiently constrct a list of valid differentials for the TEA F-function recursively. We use these conditions in tea_f_add_pddt_i to discard invalid entries early in the recursion.

To perform the checks, the following relations are used:

$dc = (da \gg R) \implies dc \in \{dc_0, dc_1, dc_2, dc_3\}$ where:

- $dc_0 = (da \gg R)$.
- $dc_2 = (da \gg R) - 2^{n-R}$.
- $dc_1 = (da \gg R) + 1$.
- $dc_3 = (da \gg R) - 2^{n-R} + 1$.

Depending on the bit position k (some of) the following checks are performed:

1. If $(k \geq R)$ perform check on the $(k-R)$ LS bits. If $(k >= R)$ we check if the first $(k-R)$ LSB bits of $(da \gg R)$ are equal to the first $(k-R)$ bits of $dc_i,\ 0 \leq i < 4$ according to the above equations. So we check if any of the following four equations hold:
   - $(da \gg R)[0 : (k-R)] = (dc_0)[0 : (k-R)]$.
   - $(da \gg R)[0 : (k-R)] = (dc_0 + 2^{n-R})[0 : (k-R)]$.
   - $(da \gg R)[0 : (k-R)] = (dc_0 - 1)[0 : (k-R)]$.
   - $(da \gg R)[0 : (k-R)] = (dc_0 + 2^{n-R} - 1)[0 : (k-R)]$.
2. Check that the R LS bits of da are not zero $da[(r-1) : 0] \neq 0$.
3. If $(k >= R) \wedge (k > (n-R))$ check the $(n-R)$ MS bits. When $(k > (n-R))$, $(da \gg R)[k] = 0$ and we check the top $(n-R)$ MS bits of dc. More specifically, we check if the initial four equations hold for the $(n-R)$ MS bits of the operands:
   - $dc_0[(n-1) : (n-R+1)] = (da \gg R)[(n-1) : (n-R+1)]$.
   - $dc_1[(n-1) : (n-R+1)] = ((da \gg R) + 1)[(n-1) : (n-R+1)]$.
   - $dc_2[(n-1) : (n-R+1)] = ((da \gg R) - 2^{n-R})[(n-1) : (n-R+1)]$.
   - $dc_3[(n-1) : (n-R+1)] = ((da \gg R) - 2^{n-R} + 1)[(n-1) : (n-R+1)]$.

7.20.2.3 void tea_f_add_pddt (uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::set< differential_t, struct_comp_diff_dx_dy > * *diff_set_dx_dy*)

Compute a partial DDT (pDDT) for the TEA F-function: wrapper function of tea_f_add_pddt_i . By definition a pDDT contains only differentials that have probability above a fixed probability thershold.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is TEA_ADD_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_set_dx_-dy	set of differentials $(dx \to dy)$ in the pDDT ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

tea_f_add_pddt_i.

7.20.2.4 void tea_f_add_pddt_adjust_to_key (uint32_t *nrounds*, uint32_t *npairs*, uint32_t *key[4]*, double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy*)

Adjust the probabailities of the differentials in a pDDT computed with tea_f_add_pddt , to the value of a fixed key by performing one-round TEA encryptions over a number of chosen plaintext pairs drawn uniformly at random.

Parameters

nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
key	cryptographic key of TEA.
p_thres	probability threshold (TEA_ADD_P_THRES).
diff_set_dx_-dy	set of differentials (the pDDT) ordered by index $i = (dx\,2^n + dy)$ - small-est first.

7.20.2.5 void tea_f_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > ∗ *diff_mset_p*, const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

From a pDDT represented in the form of a set of differentials ordered by index, compute a pDDT as a set of differentials ordered by probability.

Parameters

diff_mset_p	output pDDT: set of differentials $(dx \to dy)$ ordered by probability; stored in an STL multiset structure, internally implemented as a Red--Black binary search tree.
diff_set_dx_-dy	input pDDT: set of differentials $(dx \to dy)$ ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

7.20.2.6 void tea_f_add_pddt_exper (gsl_matrix ∗ *A[2][2][2][2]*, uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset< differential_t, struct_comp_diff_p > ∗ *diff_mset_p*)

Experimentally compute the full DDT of the TEA F-function containining expected probabilities, averaged over all keys and round constants. An exhautive search is performed over all input and output differences. **Complexity:** $O(2^{2n})$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is TEA_ADD_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	set of differentials $(dx \rightarrow dy)$ ordered by probability (the DDT).

7.20.2.7 void tea_f_add_pddt_fk_exper (uint32_t *n*, double *p_thres*, uint32_t *delta*, uint32_t *k0*, uint32_t *k1*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset< differential_t, struct_comp_diff_p > ∗ *diff_mset_p*)

Experimentally compute the full DDT of the TEA F-function containining probabilities for a fixed key and round constant. An exhautive search is performed over all input and output differences. **Complexity:** $O(2^{2n})$.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is TEA_ADD_P_THRES).
delta	round constant.
k0	first round key.
k1	second round key.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	set of differentials $(dx \rightarrow dy)$ ordered by probability (the DDT).

7.20.2.8 void tea_f_add_pddt_i (const uint32_t *k*, const uint32_t *n*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, gsl_matrix ∗ *A[2][2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, uint32_t ∗ *dd*, double ∗ *p*, const double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy*)

Computes a partial difference distribution table (pDDT) for the F-function of block cipher TEA.

Parameters

k	current bit position in the recursion.
n	word size (default is WORD_SIZE).
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
C	unit column vector for computing $\mathrm{adp}^{3\oplus}$ (adp_xor3).
da	first input difference to the XOR operation in F.
db	second input difference to the XOR operation in F.
dc	third input difference to the XOR operation in F.
dd	output difference from the XOR operation in F.
p	probability of the partially constructed differential $(da[k:0], db[k:0], dc[k:0] \rightarrow dd[k:0])$.
p_thres	probability threshold (default is TEA_ADD_P_THRES).
diff_set_dx_-dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Attention

The computed pDDT is based on the expected additive differential probability of the TEA F-function (eadp_tea_f), averaged over all round keys and round constants δ and therefore contains average (as opposed to fixed-key fixed-constants adp_tea-_f_fk) probabilities.

Algorithm Outline:

Applies conceptually the same logic as adp_xor_pddt_i. It recursively constructs all differentials for the XOR operation with three inputs $(da, db, dc \rightarrow dd)$, with the additional requirement that they must satisfy the following properties:

1. $\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd) > p_{\mathrm{thres}}$.
2. $db = da \ll 4$.
3. $dc \in (da \ll R), (da \ll R) + 1, (da \ll R) - 2^{n-R}, (da \ll R) - 2^{n-R} + 1$, so that $dc = (da \ll R)$ where $R =$TEA_RSH_CONST.

Only the entries for which $\mathrm{eadp}^F(da \rightarrow dd) > p_{\mathrm{thres}}$ are stored.

See also

adp_xor_pddt_i, lsh_condition_is_sat, rsh_condition_is_sat.

7.21 include/tea.hh File Reference

Header file for tea.cc. Common functions used in the analysis of TEA. .

Defines

- #define TEA_ADD_P_THRES 0.05
- #define TEA_ADD_MAX_PDDT_SIZE (1U << 20)
- #define TEA_NCYCLES 32

Functions

- void tea_encrypt (uint32_t ∗v, uint32_t ∗k, int nrounds)
- uint32_t tea_f (uint32_t x, uint32_t k0, uint32_t k1, uint32_t delta, uint32_t lsh_-const, uint32_t rsh_const)
- uint32_t tea_f_i (const uint32_t mask_i, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t x_in)
- void tea_compute_delta_const (uint32_t D[TEA_NCYCLES])
- double **tea_add_diff_adjust_to_key** (const uint64_t npairs, const int round_idx, const uint32_t da, const uint32_t db, const uint32_t key[4])
- double **tea_differential_thres_exper_fk** (uint64_t npairs, int r, uint32_t key[4], uint32_t da[2], uint32_t db[2])
- uint32_t **tea_add_verify_trail** (uint32_t nrounds, uint32_t npairs, uint32_t key[4], differential_t trail[NROUNDS])
- uint32_t **tea_add_verify_differential** (uint32_t nrounds, uint32_t npairs, uint32_t key[4], differential_t trail[NROUNDS])
- void **print_trail_latex** (FILE ∗fp, uint32_t nrounds, uint32_t keys[4], differential_t trail[NROUNDS])

7.21.1 Detailed Description

Header file for tea.cc. Common functions used in the analysis of TEA. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.21.2 Define Documentation

7.21.2.1 #define TEA_ADD_MAX_PDDT_SIZE (1U << 20)

Maximum size of the pDDT for ADD differences.

7.21.2.2 #define TEA_ADD_P_THRES 0.05

Probability threshold for ADD differences.

7.21.2.3 #define TEA_NCYCLES 32

Cycles in TEA: 1 cycle = 2 rounds.

7.21.3 Function Documentation

7.21.3.1 void tea_compute_delta_const (uint32_t *D[TEA_NCYCLES]*)

Compute all round constants of block cipher TEA.

Parameters

D	all round constants δ of TEA.

7.21.3.2 void tea_encrypt (uint32_t $*$ *v*, uint32_t $*$ *k*, int *nrounds*)

Round-reduced version of block cipher TEA. Reference: https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm.

Parameters

v	plaintext.
k	secret key.
nrounds	number of rounds ($1 \leq$ `nrounds` ≤ 64).

7.21.3.3 uint32_t tea_f (uint32_t *x*, uint32_t *k0*, uint32_t *k1*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

The F-function of block cipher TEA: $F(x) = ((x \ll 4) + k_0) \oplus (x + \delta) \oplus ((x \gg 5) + k_1)$.

Parameters

x	input to F.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$F(x)$

7.21.3.4 uint32_t tea_f_i (const uint32_t *mask_i*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *x_in*)

The F-function of block cipher TEA (tea_f) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
k0	first round key.
k1	second round key.

delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	input to F.

Returns

$F(x) \bmod 2^i$

Attention

the initial value `x_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f()

7.22 include/xdp-add-pddt.hh File Reference

Header file for xdp-add-pddt.cc. Compute a partial difference distribution table (pDDT) for xdp^{+}. .

Functions

- uint32_t xdp_add_pddt_exper (std::multiset< differential_3d_t, struct_comp_diff_3d_p > ∗diff_set, double p_thres)
- void xdp_add_pddt_i (const uint32_t k, const uint32_t n, const double p_thres, gsl_matrix ∗A[2][2][2], gsl_vector ∗C, uint32_t ∗da, uint32_t ∗db, uint32_t ∗dc, double ∗p, std::multiset< differential_3d_t, struct_comp_diff_3d_p > ∗diff_set)
- void xdp_add_pddt (uint32_t n, double p_thres)

7.22.1 Detailed Description

Header file for xdp-add-pddt.cc. Compute a partial difference distribution table (pDDT) for xdp^{+}. .

Author

V.Velichkov, `vesselin.velichkov@uni.lu`

Date

2012-2013

7.22.2 Function Documentation

7.22.2.1 void **xdp_add_pddt** (uint32_t *n*, double *p_thres*)

Compute a partial DDT for xdp^{+}: wrapper function of xdp_add_pddt_i.

Parameters

n	word size.
p_thres	probability threshold.

See also

xdp_add_pddt_i.

7.22.2.2 uint32_t **xdp_add_pddt_exper** (std::multiset< **differential_3d_t, struct_comp_diff_3d_p** > ∗ *diff_set*, double *p_thres*)

Compute a partial DDT for xdp^{+} by exhasutive search over all input and output differences.

Parameters

diff_set	set of all differentials with probability not less than the threshold (the pDDT)
p_thres	probability threshold.

Returns

number of elements in the pDDT.

See also

xdp_add_pddt_i

7.22.2.3 void **xdp_add_pddt_i** (const uint32_t *k*, const uint32_t *n*, const double *p_thres*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, double ∗ *p*, std::multiset< **differential_3d_t, struct_comp_diff_3d_p** > ∗ *diff_set*)

Recursively compute all XOR differentials $(da, db \rightarrow dc)$ for ADD that have probability xdp^{+} larger than a fixed probability threshold `p_thres`.

The function works recursively starting from the LS bit `k` = 0 and terminating at the -MS bit `n`. At every bit position i it assigns values to the i-th bits of the differences `da`, `db`, `dc` and evaluates the probability of the resulting partial (i+1)-bit differential: $(da[i:0], db[i:0] \rightarrow dc[i:0])$. The recursion proceeds only if this probability is not less than the threshold `p_thres`. When i = n, the differential $(da[n-1:0], db[n-1:0] \rightarrow dc[n-1:0])$ is stored in an STL multiset structure (internally implemented as a Red-Black tree).

The **complexity** is strongly dependent on the threshold and is worst-case exponential in the word size: $O(2^{3n})$.

Note

If `p_thres` = 0.0 then the full DDT is computed.
Can be used also to compute all differentials that have non-zero probability by setting p_thres $>$ 0.0 .
For 32 bit words, recommended values for the threshold are p_thres $>=$ 0.7.

Parameters

k	current bit position in the recursion.
n	word size.
p_thres	probability threshold.
A	transition probability matrices for xdp^{+}.
da	first input difference.
db	second input difference.
dc	output difference.
p	probability of the differential $(da[k:0], db[k:0] \rightarrow dc[k:0])$.
diff_set	set of all differentials with probability not less than the threshold (the pDDT)

7.23 include/xdp-add.hh File Reference

Header file for xdp-add.cc: The XOR differential probability of ADD $\mathrm{xdp}^{+}(da, db \rightarrow db)$.
.

Defines

- #define XDP_ADD_MSIZE 4
- #define XDP_ADD_NMATRIX 8
- #define XDP_ADD_NINPUTS 2
- #define XDP_ADD_ISTATE 0
- #define XDP_ADD_COLSUM 4
- #define XDP_ADD_NORM 1.0 /(double)XDP_ADD_COLSUM

Functions

- void xdp_add_alloc_matrices (gsl_matrix ∗A[2][2][2])
- void xdp_add_free_matrices (gsl_matrix ∗A[2][2][2])
- void xdp_add_normalize_matrices (gsl_matrix ∗A[2][2][2])
- void xdp_add_print_matrices (gsl_matrix ∗A[2][2][2])
- void xdp_add_sf (gsl_matrix ∗A[2][2][2])
- double xdp_add (gsl_matrix ∗A[2][2][2], uint32_t da, uint32_t db, uint32_t dc)
- double xdp_add_exper (const uint32_t da, const uint32_t db, const uint32_t dc)

- void xdp_add_pddt_i (const uint32_t k, const uint32_t n, const double p_thres, gsl_matrix ∗A[2][2][2], gsl_vector ∗C, uint32_t ∗da, uint32_t ∗db, uint32_t ∗dc, double ∗p, std::multiset< differential_3d_t, struct_comp_diff_3d_p > ∗diff_set)
- void **xdp_add_pddt** ()

7.23.1 Detailed Description

Header file for xdp-add.cc: The XOR differential probability of ADD $\mathrm{xdp}^{+}(da, db \rightarrow db)$.
.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.23.2 Define Documentation

7.23.2.1 #define XDP_ADD_COLSUM 4

Sum of non-zero elements in one column of the xdp^{+} matrices.

7.23.2.2 #define XDP_ADD_ISTATE 0

Initial state for computing the xdp^{+} S-function.

7.23.2.3 #define XDP_ADD_MSIZE 4

Number of state values in the xdp^{+} S-function.

7.23.2.4 #define XDP_ADD_NINPUTS 2

Number of inputs to the XOR operation.

7.23.2.5 #define XDP_ADD_NMATRIX 8

Number of xdp^{+} matrices.

7.23.2.6 #define XDP_ADD_NORM 1.0 /(double)XDP_ADD_COLSUM

Normalization factor for the xdp^{+} matrices.

7.23.3 Function Documentation

7.23.3.1 double xdp_add (gsl_matrix ∗ *A[2][2][2],* uint32_t *da,* uint32_t *db,* uint32_t *dc*)

The XOR differential probability of ADD (xdp^{+}). **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for xdp^{+} computed with xdp_add_sf.
da	first input difference.
db	second input difference.
dc	output difference.

Returns

$p = \mathrm{xdp}^{+}(da, db \rightarrow dc)$

See also

adp_xor

7.23.3.2 void xdp_add_alloc_matrices (gsl_matrix ∗ *A[2][2][2]*)

Allocate memory for the transition probability matrices for xdp^{+}.

Parameters

A	transition probability matrices for xdp^{+}.

See also

xdp_add_free_matrices

7.23.3.3 double xdp_add_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*)

The XOR differential probability of ADD (xdp^{+}) computed experimentally over all inputs. **Complexity:** $O(2^{2n})$.

Parameters

da	first input difference.
db	second input difference.
dc	output difference.

Returns

$p = \mathrm{xdp}^{+}(da, db \rightarrow dc)$

See also

xdp_add

7.23.3.4 void xdp_add_free_matrices (gsl_matrix ∗ *A[2][2][2]*)

Free memory reserved by a previous call to xdp_add_alloc_matrices.

Parameters

A	transition probability matrices for xdp^{+}.

7.23.3.5 void xdp_add_normalize_matrices (gsl_matrix * *A[2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for xdp^{+}.

7.23.3.6 void xdp_add_pddt_i (const uint32_t *k*, const uint32_t *n*, const double *p_thres*, gsl_matrix * *A[2][2][2]*, gsl_vector * *C*, uint32_t * *da*, uint32_t * *db*, uint32_t * *dc*, double * *p*, std::multiset< differential_3d_t, struct_comp_diff_3d_p > * *diff_set*)

Recursively compute all XOR differentials $(da, db \rightarrow dc)$ for ADD that have probability xdp^{+} larger than a fixed probability threshold `p_thres`.

The function works recursively starting from the LS bit `k` = 0 and terminating at the -MS bit `n`. At every bit position i it assigns values to the i-th bits of the differences `da`, `db`, `dc` and evaluates the probability of the resulting partial (i+1)-bit differential: $(da[i:0], db[i:0] \rightarrow dc[i:0])$. The recursion proceeds only if this probability is not less than the threshold `p_thres`. When i = n, the differential $(da[n-1:0], db[n-1:0] \rightarrow dc[n-1:0])$ is stored in an STL multiset structure (internally implemented as a Red-Black tree).

The **complexity** is strongly dependent on the threshold and is worst-case exponential in the word size: $O(2^{3n})$.

Note

If `p_thres` = 0.0 then the full DDT is computed.
Can be used also to compute all differentials that have non-zero probability by setting p_thres $>$ 0.0 .
For 32 bit words, recommended values for the threshold are p_thres $>=$ 0.7.

Parameters

k	current bit position in the recursion.
n	word size.
p_thres	probability threshold.
A	transition probability matrices for xdp^{+}.
da	first input difference.
db	second input difference.
dc	output difference.
p	probability of the differential $(da[k:0], db[k:0] \rightarrow dc[k:0])$.
diff_set	set of all differentials with probability not less than the threshold (the pDDT)

7.23.3.7 void xdp_add_print_matrices (gsl_matrix ∗ *A[2][2][2]*)

Print the matrices for xdp^{+}.

Parameters

A	transition probability matrices for xdp^{+}.

7.23.3.8 void xdp_add_sf (gsl_matrix ∗ *A[2][2][2]*)

S-function for xdp^{+}: $\text{xdp}^{+}(da, db \rightarrow db)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\text{xdp}^{+}(da, db \rightarrow db)$.

$A[2][2][2] = A[da[i]][db[i]][dc[i]]$, where

- $da[i]$: the i-th bit of the first input difference.
- $db[i]$: the i-th bit of the second input difference.
- $dc[i]$: the i-th bit of the output difference.

See also

adp_xor_sf

7.24 include/xdp-tea-f-fk.hh File Reference

Header file for xdp-tea-f-fk.cc. The XOR differential probability (XDP) of the F-function of TEA for a fixed key and round constants: $\text{xdp}^{F}(k_0, k_1, \delta | \, da \rightarrow dd)$. .

Functions

- double xdp_f_fk_exper (const uint32_t da, const uint32_t db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_xdp_f_fk_dx_exper (uint32_t ∗max_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32-_t rsh_const)
- double max_xdp_f_fk_dy_exper (const uint32_t dx, uint32_t ∗max_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32-_t rsh_const)
- bool xdp_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32-_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, int32_t x)

- bool xdp_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- uint32_t xdp_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double xdp_f_fk (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint32_t xdp_f_assign_bit_x_dx (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dx)
- double max_dx_xdp_f_fk (const uint32_t n, uint32_t ∗ret_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint32_t xdp_f_assign_bit_x_dy (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dy)
- double max_dy_xdp_f_fk (const uint32_t n, const uint32_t dx, uint32_t ∗ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)

7.24.1 Detailed Description

Header file for xdp-tea-f-fk.cc. The XOR differential probability (XDP) of the F-function of TEA for a fixed key and round constants: $\mathrm{xdp}^F(k_0,k_1,\delta|\ da \rightarrow dd)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.24.2 Function Documentation

7.24.2.1 double max_dx_xdp_f_fk (const uint32_t *n,* uint32_t ∗ *ret_dx,* const uint32_t *dy,* const uint32_t *k0,* const uint32_t *k1,* const uint32_t *delta,* const uint32_t *lsh_const,* const uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{xdp}^F(k_0,k_1,\delta|\ dx \rightarrow dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
ret_dx	maximum probability input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{xdp}^{F}(k_0, k_1, \delta |\ dx \rightarrow dy)$.

See also

xdp_f_assign_bit_x_dx

7.24.2.2 double max_dy_xdp_f_fk (const uint32_t *n*, const uint32_t *dx*, uint32_t ∗ *ret_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{xdp}^{F}(k_0, k_1, \delta |\ dx \rightarrow dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory requirement:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dy} \mathrm{xdp}^{F}(k_0, k_1, \delta |\ dx \rightarrow dy)$.

See also

xdp_f_assign_bit_x_dy, max_dy_xdp_f_fk

7.24.2.3 double max_xdp_f_fk_dx_exper (uint32_t * *max_dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{xdp}^{F}(k_0, k_1, \delta|\ dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

max_dx	maximum probability input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{xdp}^{F}(k_0, k_1, \delta|\ dx \rightarrow dy)$.

See also

max_dx_xdp_f_fk

7.24.2.4 double max_xdp_f_fk_dy_exper (const uint32_t *dx*, uint32_t * *max_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{xdp}^{F}(k_0, k_1, \delta|\ dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx	input difference.
max_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

max_dy_xdp_f_fk

7.24.2.5 uint32_t xdp_f_assign_bit_x (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the F-function of TEA is satisfied. The function operates by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i + rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key XOR probability of `F`: $\mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_f_fk

7.24.2.6 uint32_t xdp_f_assign_bit_x_dx (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dx*)

For given output difference `dy`, compute all input differences `dx` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability input difference.

The function works by recursively assigning the bits of `x` and `dx` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_f_is_sat .

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dx`.
ret_prob	the maximum probability over all input differences $\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta \vert\, dx \rightarrow dy)$.
ret_dx	the input difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_f_assign_bit_x, max_dx_xdp_f_fk

7.24.2.7 uint32_t xdp_f_assign_bit_x_dy (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t ∗ *x_cnt*, double ∗ *ret_prob*, uint32_t ∗ *ret_dy*)

For given input difference `dx`, compute all output differences `dy` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability output difference.

The function works by recursively assigning the bits of `x` and `dy` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.

mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dy`.
ret_prob	the maximum probability over all output differences $\max_{dy} \mathrm{xdp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dy	the output difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_f_assign_bit_x_dx

7.24.2.8 bool **xdp_f_check_x** (const uint32_t *lsh_const,* const uint32_t *rsh_const,* const uint32_t *k0,* const uint32_t *k1,* const uint32_t *delta,* const uint32_t *dx,* const uint32_t *dy,* const uint32_t *x*)

Check if a given value `x` satisfies the XOR differential $(dx \rightarrow dy)$ for the TEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k_0, k_1, \delta: \; dy = F(x \oplus dx) \oplus F(x)$.

7.24.2.9 double xdp_f_fk (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher TEA: $\mathrm{xdp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$.

See also

xdp_f_assign_bit_x

7.24.2.10 double xdp_f_fk_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher TEA: $\mathrm{xdp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

xdp_f_fk

7.24.2.11 bool xdp_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, int32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` is satisfied on the `i` LS bits of `x` i.e. check if $k_0, k_1, \delta : \; dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0])$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i + rsh_const`).

Returns

TRUE if $k_0, k_1, \delta : \; dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0)$.

7.25 include/xdp-xtea-f-fk.hh File Reference

Header file for xdp-xtea-f-fk.cc. The XOR differential probability (XDP) of the F-function of XTEA for a fixed key and round constants: $\mathrm{xdp}^F(k, \delta | \, da \rightarrow dd)$. .

Functions

- double xdp_xtea_f_fk_exper (const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double xdp_xtea_f_fk_approx (const uint32_t ninputs, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)

- bool xdp_xtea_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool xdp_xtea_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- uint32_t xdp_xtea_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double xdp_xtea_f_fk (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double xdp_xtea_f2_fk_exper (const uint32_t daa, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double xdp_xtea_f2_fk_approx (const uint32_t ninputs, const uint32_t daa, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- bool xdp_xtea_f2_check_x_xx (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dxx, const uint32_t dx, const uint32_t dy, const uint32_t xx, const uint32_t x)
- bool xdp_xtea_f2_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dxx, const uint32_t dx, const uint32_t dy, const uint32_t xx, const uint32_t x)
- uint32_t xdp_xtea_f2_assign_bit_x_xx (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t xx, const uint32_t x, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dxx, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗prob)
- double xdp_xtea_f2_fk (const uint32_t n, const uint32_t dxx, const uint32_t dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double nz_xdp_xtea_f (gsl_matrix ∗A[2][2][2], const uint32_t dx, uint32_t ∗dy, uint32_t lsh_const, uint32_t rsh_const)

7.25.1 Detailed Description

Header file for xdp-xtea-f-fk.cc. The XOR differential probability (XDP) of the F-function of XTEA for a fixed key and round constants: $\mathrm{xdp}^F(k,\delta|\,da \rightarrow dd)$. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.25.2 Function Documentation

7.25.2.1 double nz_xdp_xtea_f (gsl_matrix * *A[2][2][2]*, const uint32_t *dx*, uint32_t * *dy*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For the XTEA F-function (xtea_f), for fixed input difference `dx`, compute an output difference `dy` such that the differential $(dx \rightarrow dy)$ has non-zero probability.

Parameters

A	transition probability matrices for xdp^{+} (xdp_add_sf).
dx	input difference.
dy	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k, \delta | \, dx \rightarrow dy)$

Algorithm sketch:

1. Compute the output XOR difference after xtea_f_lxr : $dx_{\mathrm{LXR}} = (((dx \ll 4) \oplus (dx \gg 5))$.
2. Compute the maximum probability output difference dy after the modular addition of xtea_f : $p_{\max} = \max_{dy} \mathrm{xdp}^{+}(dx, dx_{\mathrm{LXR}} \rightarrow dy)$ (see max_xdp_add).
3. Store dy and return $p_{\max}$.

Attention

In the computation of $\max_{dy}$ xdp the inputs to the addition are implicitly assumed to be independent. Clearly they are not and so the returned probability is only an approximation.

7.25.2.2 uint32_t xdp_xtea_f2_assign_bit_x_xx (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *xx*, const uint32_t *x*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *prob*)

Counts the number of values `xx` and `x` for which the XOR differential $(dxx, dx \rightarrow dy)$ of the XTEA F-function with two inputs $F'(xx, x) = xx + F(x)$ (see xtea_f2) is satisfied. The algorithm operates by recursively assigning the bits of `xx` and `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_xtea_f2_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
xx	first input value.
x	second input value of size at least (`i + rsh_const`).
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
dxx	first input difference.
dx	second input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key XOR probability of `F'` : $\mathrm{xdp}^{F'}(k, \delta \mid dx \rightarrow dy)$.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_xtea_f2_fk

Note

x_cnt counts both the values for x and for xx.

7.25.2.3 bool xdp_xtea_f2_check_x_xx (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *xx*, const uint32_t *x*)

Check if given input values `xx` and `x` satisfy the XOR differential $(dxx, dx \rightarrow dy)$ of the XTEA F-function with two inputs $F'(xx, x) = xx + F(x)$ (see xtea_f2).

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dxx	first input difference.
dx	second input difference.
dy	output difference.
xx	first input value.
x	second input value.

Returns

TRUE if $k, \delta: \ dy = F'(xx \oplus dxx, \ x \oplus dx) \oplus F'(xx, \ x)$.

7.25.2.4 double xdp_xtea_f2_fk (const uint32_t *n*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of of the XTEA F-function with two inputs $F'(xx,x) = xx + F(x)$ (see xtea_f2): $\mathrm{xdp}^{F}(k, \delta | \ dxx, dx \rightarrow dy)$.
Complexity: $O(n) < c \leq O(2^{2n})$.

Parameters

n	word size.
dxx	first input difference.
dx	second input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^{F'}(k, \delta | \ dxx, dx \rightarrow dy)$.

See also

xdp_xtea_f2_assign_bit_x_xx

7.25.2.5 double xdp_xtea_f2_fk_approx (const uint32_t *ninputs*, const uint32_t *daa*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the XDP of the XTEA F-function with two inputs $F'(xx,x) = xx + F(x)$ (see xtea_f2), obtained over a number of input chosen plaintext pairs c hosen uniformly at random.

Parameters

ninputs	number of input chosen plaintext pairs.
daa	first input difference.
da	second input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^{F'}(k,\delta|\ daa,da \rightarrow dy)$.

7.25.2.6 double xdp_xtea_f2_fk_exper (const uint32_t *daa*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key through exhaustive search over all input values the fixed-constant XOR differential probability of the F-function of block cipher XTEA including the second modular addition and denoted by $F'(xx,x) = xx + F(x)$ (see xtea_f2): $\mathrm{xdp}^{F'}(k,\delta|\ dxx,dx \rightarrow dy)$. **Complexity:** $O(2^{2n})$.

Parameters

daa	first input difference.
da	second input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^{F'}(k,\delta|\ daa,da \rightarrow dy)$.

7.25.2.7 bool xdp_xtea_f2_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *xx*, const uint32_t *x*)

Check if the XOR differential $(dxx,dx \rightarrow dy)$ of the XTEA F-function with two inputs $F'(xx,x) = xx + F(x)$ (see xtea_f2) is satisfied on the `i` LS bits of `xx` and `x` i.e. check if

$$k,\delta:\ dy[i-1:0] = F(xx[i-1:0], x[i-1:0] \oplus dx[i-1:0]) \oplus F(xx[i-1:0], x[i-1:0])$$

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dxx	first input difference.
dx	second input difference.
dy	output difference.
xx	first input value.
x	second input value of size at least (`i + rsh_const`).

Returns

TRUE if the differential is satisfied; FALSE otherwise.

Attention

x must be of size at least $(i+R)$ bits where R is the RSH constant of F.

7.25.2.8 uint32_t **xdp_xtea_f_assign_bit_x** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values x for which the differential $(dx \rightarrow dy)$ for the F-function of XTEA is satisfied. The function operates by recursively assigning the bits of x starting from bit position i and terminating at the MS bit n. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the i LS bits. This is checked by applying xdp_xtea_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the i LS bits of x.
x	input value of size at least (i + rsh_const).
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key XOR probability of F: $\mathrm{xdp}^F(k, \delta \mid dx \rightarrow dy)$.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

xdp_xtea_f_fk

7.25.2.9 bool **xdp_xtea_f_check_x** (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value x satisfies the XOR differential $(dx \rightarrow dy)$ for the XTEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k, \delta : \; dy = F(x \oplus dx) \oplus F(x)$.

7.25.2.10 double xdp_xtea_f_fk (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher XTEA: $\mathrm{xdp}^F(k, \delta | \; dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k, \delta | \; dx \rightarrow dy)$.

See also

xdp_xtea_f_assign_bit_x

7.25.2.11 double xdp_xtea_f_fk_approx (const uint32_t *ninputs*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the XDP of the XTEA F-function (xtea_f) obtained over a number of input chosen plaintext pairs chosen uniformly at random.

Parameters

ninputs	number of input chosen plaintext pairs.
da	input difference.

db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k,\delta|\ dx \rightarrow dy)$.

7.25.2.12 double xdp_xtea_f_fk_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher XTEA: $\mathrm{xdp}^F(k,\delta|\ dx \rightarrow dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k,\delta|\ dx \rightarrow dy)$.

7.25.2.13 bool xdp_xtea_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` (xtea_f) is satisfied on the `i` LS bits of `x` i.e. check if

$k,\delta:\ dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0])$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.

k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i + rsh_const`).

Returns

TRUE if $k,\delta:\ dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0])$.

7.26 include/xtea-add-threshold-search.hh File Reference

Header file for xtea-add-threshold-search.cc. Automatic search for ADD differential trails in block cipher XTEA. .

Functions

- void xtea_add_threshold_search (const int n, const int nrounds, const uint32_t npairs, const uint32_t round_key[64], const uint32_t round_delta[64], gsl_matrix *A[2][2][2], gsl_matrix *AA[2][2][2], double B[NROUNDS], double *Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS], uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void xtea_add_trail_search (uint32_t key[4], uint32_t round_key[64], uint32_t round_delta[64])

7.26.1 Detailed Description

Header file for xtea-add-threshold-search.cc. Automatic search for ADD differential trails in block cipher XTEA. .

Author

V.Velichkov, `vesselin.velichkov@uni.lu`

Date

2012-2013

7.26.2 Function Documentation

7.26.2.1 void xtea_add_threshold_search (const int *n*, const int *nrounds*, const uint32_t *npairs*, const uint32_t *round_key[64]*, const uint32_t *round_delta[64]*, gsl_matrix $*$ *A[2][2][2]*, gsl_matrix $*$ *AA[2][2][2]*, double *B[NROUNDS]*, double $*$ *Bn*, const differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset$<$ differential_t, struct_comp_diff_p $>*$ *diff_mset_p*, std::set$<$ differential_t, struct_comp_diff_dx_dy $>*$ *diff_set_dx_dy*)

Automatic search for ADD differential trails in block cipher XTEA using pDDT.

Note

For more details on the algorithm see tea_add_threshold_search.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.
A	transition probability matrices for $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for XOR with fixed input $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ (adp_xor_fixed_input_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best found probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best found differential trail for `nrounds`.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	set of differentials (dx, dy, p) (the pDDT) ordered by probability p.
diff_set_dx_dy	set of differentials (dx, dy, p) (the pDDT) ordered by index $i = (dx\, 2^n + dy)$.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

See also

tea_add_threshold_search.

7.26.2.2 void xtea_add_trail_search (uint32_t *key[4]*, uint32_t *round_key[64]*, uint32_t *round_delta[64]*)

Search for ADD differential trails in block cipher XTEA: wrapper function for tea_add_threshold_search.

Parameters

key	cryptographic key of XTEA.
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.

Algorithm Outline:

The procedure operates as follows:

1. Compute a pDDT for F (xtea_f_add_pddt).
2. Adjust the probabilities of the pDDT to the round key and constant (adp_xtea_f_approx).
3. Execute the search for differential trails for n rounds (n = NROUNDS) through a successive application of xtea_add_threshold_search :
    - Compute the best found probability on 1 round: $B[0]$.
    - Using $B[0]$ compute the best found probability on 2 rounds: $B[1]$.
    - ...
    - Using $B[0],\ldots,B[i-1]$ compute the best found probability on $(i+1)$ rounds: $B[i]$.
    - ...
    - Using $B[0],\ldots,B[n-2]$ compute the best found probability on n rounds: $B[n-1]$.
4. Print the best found trail on n rounds on standrad output and terminate.

See also

xtea_add_threshold_search, tea_add_trail_search

7.27 include/xtea-f-add-pddt.hh File Reference

Declarations for xtea-f-add-pddt.cc. Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher XTEA. .

Functions

- void xtea_f_add_pddt_i (const uint32_t k, const uint32_t n, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, gsl_matrix ∗A[2][2][2], gsl_matrix ∗AA[2][2][2], gsl_vector ∗C, uint32_t ∗da, uint32_t ∗db, uint32_t ∗dc, uint32_t ∗dd, double ∗p, const double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy)
- void xtea_f_add_pddt (uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, gsl_matrix ∗A[2][2][2], gsl_matrix ∗AA[2][2][2], gsl_vector ∗C, uint32_t key, uint32_t delta, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy)
- void xtea_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > ∗diff_mset_p, const std::set< differential_t, struct_comp_diff_dx_dy > diff_set_dx_dy)

7.27.1 Detailed Description

Declarations for xtea-f-add-pddt.cc. Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher XTEA. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.27.2 Function Documentation

7.27.2.1 void xtea_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > ∗ *diff_mset_p*, const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

From a pDDT represented in the from of a set of differentials ordered by index, compute a pDDT as a set of differentials ordered by probability.

Parameters

diff_mset_p	output pDDT: set of differentials $(dx \rightarrow dy)$ ordered by probability; stored in an STL multiset structure, internally implemented as a Red--Black binary search tree.
diff_set_dx_dy	input pDDT: set of differentials $(dx \rightarrow dy)$ ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

tea_f_add_pddt_dxy_to_dp

7.27.2.2 void xtea_f_add_pddt (uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, gsl_matrix $*$ *A[2][2][2]*, gsl_matrix $*$ *AA[2][2][2]*, gsl_vector $*$ *C*, uint32_t *key*, uint32_t *delta*, std::set< differential_t, struct_comp_diff_dx_dy > $*$ *diff_set_dx_dy*)

Compute a partial DDT (pDDT) for the XTEA F-function: wrapper function of xtea_f_add_pddt_i . By definition a pDDT contains only differentials that have probability above a fixed probability thershold.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is XTEA_ADD_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
A	transition probability matrices for $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for XOR with fixed input $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ (adp_xor_fixed_input_sf).
C	unit column vector for computing $\mathrm{adp}^{\oplus}$ (adp_xor).
key	round key.
delta	round constant.
diff_set_dx_dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

tea_f_add_pddt_i.

7.27.2.3 void xtea_f_add_pddt_i (const uint32_t *k*, const uint32_t *n*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, gsl_matrix $*$ *A[2][2][2]*, gsl_matrix $*$ *AA[2][2][2]*, gsl_vector $*$ *C*, uint32_t $*$ *da*, uint32_t $*$ *db*, uint32_t $*$ *dc*, uint32_t $*$ *dd*, double $*$ *p*, const double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > $*$ *diff_set_dx_dy*)

Computes an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA.

Parameters

k	current bit position in the recursion.
n	word size (default is WORD_SIZE).
key	round key.
delta	round constant.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).

A	transition probability matrices for XOR $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for XOR with fixed input $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ (adp_xor-_fixed_input_sf).
C	unit column vector for computing $\mathrm{adp}^{\oplus}$ (adp_xor).
da	input difference to the F-function of XTEA.
db	output difference from the LSH operation in F.
dc	output difference from the RSH operation in F.
dd	output difference from the XOR operation in F.
p	probability of the partially constructed differential $(db[k:0], dc[k:0] \rightarrow dd[k:0])$ for the XOR operation in F.
p_thres	probability threshold (default is XTEA_ADD_P_THRES).
diff_set_dx_-dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Algorithm Outline:

1. Recursively construct all differentials for the XOR operation in the f_{LXR} component of the F-function of XTEA (see xtea_f_lxr): $f_{\mathrm{LXR}}(a) = (((a \ll 4) \oplus (a \gg 5))$. Note that when doing this, we treat the two inputs $(a \ll 4)$ and $(a \gg 5)$ as independent inputs, denoted respectively by b and c. At every bit position in the recursion we require the corresponding partially constructed input differences da, db, dc and the output difference dd to satisfy conditions lsh_condition_is_sat and rsh_condition_is_sat. As a result, after the MSB is processed and $k = n$ the so constructed differences satisfy the following constions (see tea_f_add_pddt_i):
   (a) $\mathrm{adp}^{3\oplus}(db, dc \rightarrow dd) > p_{\mathrm{thres}}$.
   (b) $db = da \ll 4$.
   (c) $dc \in (da \ll R), (da \ll R) + 1, (da \ll R) - 2^{n-R}, (da \ll R) - 2^{n-R} + 1,$ so that $dc = (da \ll R)$ where $R =$TEA_RSH_CONST.
2. Set $dz = da + dd$ according to the feed-forward operation in F (see xtea_f) and compute the maximum probability output difference dy for the ADD operation with round key and δ (see xtea_f) with one fixed input: $\max\ \mathrm{adp}^{\oplus}_{\mathrm{FI}}((\mathrm{key} + \delta),\ dz \rightarrow dy)$.
3. Experimentally adjust the probability of the differential $\mathrm{adp}^{F}(da \rightarrow dy)$ to the full function F using adp_xtea_f_approx . Set the adjusted probability to $\hat{p}$.
4. Store $(da, dy, \hat{p})$ in the pDDT.

See also

tea_f_add_pddt_i

7.28 include/xtea-f-xor-pddt.hh File Reference

Header for xtea-f-xor-pddt.cc. Computing an XOR partial difference distribution table (pDDT) for the F-function of block cipher XTEA. .

Functions

- void xtea_f_xor_pddt_i (const uint32_t k, const uint32_t n, const uint32_t lsh-_const, const uint32_t rsh_const, gsl_matrix *A[2][2][2], gsl_vector *C, uint32-_t *da, uint32_t *db, uint32_t *dc, double *p, const double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void xtea_f_xor_pddt (uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void xtea_xor_pddt_adjust_to_key (uint32_t nrounds, uint32_t npairs, uint32_t lsh_const, uint32_t rsh_const, uint32_t key, uint32_t delta, double p_thres, std-::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void xtea_xor_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p, const std::set< differential_t, struct_comp_diff_dx_dy > diff_-set_dx_dy)

7.28.1 Detailed Description

Header for xtea-f-xor-pddt.cc. Computing an XOR partial difference distribution table (pDDT) for the F-function of block cipher XTEA. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.28.2 Function Documentation

7.28.2.1 void xtea_f_xor_pddt (uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::set< differential_t, struct_comp_diff_dx_dy > * *diff_set_dx_dy*)

Compute an XOR partial DDT (pDDT) for the XTEA F-function: wrapper function of xtea_f_xor_pddt_i . By definition a pDDT contains only differentials that have probability above a fixed probability thershold.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is XTEA_XOR_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_set_dx_-dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Note

The compuation of the pDDT is based on the ADD operation in the XTEA F-function: the only non-linear componenet with respect to XOR differences.

See also

xtea_f_xor_pddt_i.

7.28.2.2 void xtea_f_xor_pddt_i (const uint32_t *k*, const uint32_t *n*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, double ∗ *p*, const double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy*)

Computes an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA.

Parameters

k	current bit position in the recursion.
n	word size (default is WORD_SIZE).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
A	transition probability matrices for ADD xdp^+ (xdp_add_sf).
C	unit column vector for computing xdp^+ (xdp_add).
da	input difference to the F-function of XTEA.
db	output difference from the f_{LXR} component of F ((xtea_f_lxr)).
dc	output difference from the F-function of XTEA.
p	probability of the partially constructed differential $(da[k:0], db[k:0] \rightarrow dc[k:0])$ for the ADD operation in F.
p_thres	probability threshold (default is XTEA_XOR_P_THRES).
diff_set_dx_-dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Algorithm Outline:

1. Treat the two inputs to the ADD operation: a and $b = ((a << 4)^(a >> 5))$ as independent.
2. Recursively construct a list of differentials $(da, db \rightarrow dc)$ for the ADD operation in F with probability bigger than p_{thres} (see xdp_add_pddt_i).
3. Of the constructed differentials store in an pDDT only those for which it holds $db = (da \ll 4) \oplus (da \gg 5)$.
4. Return pDDT.

See also

xtea_f_xor_pddt

7.28.2.3 void xtea_xor_pddt_adjust_to_key (uint32_t *nrounds*, uint32_t *npairs*, uint32_t *lsh_const*, uint32_t *rsh_const*, uint32_t *key*, uint32_t *delta*, double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy*)

Adjust the probabailities of the differentials in a pDDT computed with xtea_f_xor_pddt , to the value of a fixed key by performing one-round TEA encryptions over a number of chosen plaintext pairs drawn uniformly at random.

Parameters

nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
key	round key.
delta	round constant.
p_thres	probability threshold (XTEA_XOR_P_THRES).
diff_set_dx_dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\,2^n + dy)$.

7.28.2.4 void xtea_xor_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > ∗ *diff_mset_p*, const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

From a pDDT represented in the from of a set of differentials ordered by index, compute a pDDT as a set of differentials ordered by probability.

Parameters

diff_mset_p	output pDDT: set of differentials $(dx \rightarrow dy)$ ordered by probability; stored in an STL multiset structure, internally implemented as a Red--Black binary search tree.
diff_set_dx_dy	input pDDT: set of differentials $(dx \rightarrow dy)$ ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

xtea_add_pddt_dxy_to_dp

7.29 include/xtea-xor-threshold-search.hh File Reference

Header file for xtea-xor-threshold-search.cc. Automatic search for XOR differential trails in block cipher XTEA. .

Functions

- double xtea_xor_init_estimate (uint32_t next_round, uint32_t lsh_const, uint32_t rsh_const, uint32_t npairs, gsl_matrix ∗A[2][2][2], double B[NROUNDS], differential_t trail[NROUNDS], std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy, uint32_t round_key[64], uint32_t round_delta[64])
- void xtea_xor_threshold_search (const int n, const int nrounds, const uint32_t npairs, const uint32_t round_key[64], const uint32_t round_delta[64], gsl_matrix ∗A[2][2][2], double B[NROUNDS], double ∗Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS], uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > ∗diff_mset_p, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy, uint32_t dxx_init, uint32_t ∗dxx_init_in)
- void xtea_xor_trail_search (uint32_t key[4], uint32_t round_key[64], uint32_t round_delta[64])

7.29.1 Detailed Description

Header file for xtea-xor-threshold-search.cc. Automatic search for XOR differential trails in block cipher XTEA. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.29.2 Function Documentation

7.29.2.1 double xtea_xor_init_estimate (uint32_t *next_round,* uint32_t *lsh_const,* uint32_t *rsh_const,* uint32_t *npairs,* gsl_matrix ∗ *A[2][2][2],* double *B[NROUNDS],* differential_t *trail[NROUNDS],* std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy,* uint32_t *round_key[64],* uint32_t *round_delta[64]*)

Compute an initial estimate of the probability of a differential trail on $(n+1)$ rounds, by greedily extending the best found trail for n rounds.

Parameters

next_round	index of round $(n+1)$ to which a trail on n rounds will be extended.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
npairs	number of chosen plaintext pairs (NPAIRS).
A	transition probability matrices for xdp^{+} (xdp_add_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
trail	best found differential trail for n rounds.

diff_set_dx_-dy	pDDT as a set of differentials (dx, dy, p) ordered by index $i = (dx\ 2^n + dy)$.
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.

See also

xtea_xor_trail_search

7.29.2.2 void **xtea_xor_threshold_search** (const int *n*, const int *nrounds*, const uint32_t *npairs*, const uint32_t *round_key[64]*, const uint32_t *round_delta[64]*, gsl_matrix $*$ *A[2][2][2]*, double *B[NROUNDS]*, double $*$ *Bn*, const **differential_t** *diff_in[NROUNDS]*, **differential_t** *trail[NROUNDS]*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset$<$ **differential_t, struct_comp_diff_p** $>$ $*$ *diff_mset_p*, std::set$<$ **differential_t, struct_comp_diff_dx_dy** $>$ $*$ *diff_set_dx_dy*, uint32_t *dxx_init*, uint32_t $*$ *dxx_init_in*)

Automatic search for XOR differential trails in block cipher TEA. using pDDT.

Parameters

n	index of the current round: $0 \leq n < \text{nrounds}$.
nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.
A	transition probability matrices for xdp^+ (xdp_add_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best found probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best found differential trail for `nrounds`.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	pDDT as a set of differentials (dx, dy, p) ordered by probability p.
diff_set_dx_-dy	pDDT as a set of differentials (dx, dy, p) ordered by index $i = (dx\ 2^n + dy)$.
dxx_init	initial left input difference to XTEA
dxx_init_in	the initial left input difference to XTEA corresponding to the best found trail (initialized to dxx_init and updated dynamically).

Attention

The pDDT contains differentials and their probabilities for the XTEA F-function F (xtea_f) as opposed to the function F' (xtea_f2) that also includes the second ADD operation. In other words, the pDDT does *not* take into account the differential probabilities arising from the second ADD operation. The latter are computed during the search.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

More Details

The differential probability (DP) for one round of XTEA is computed as the product of the DP of F (xtea_f) and the DP of the modular addition in F' (xtea_f2). The functions F and F' are defined as: $F(x) = y = x + ((x \ll 4) \oplus (x \gg 5))$, $F'(xx, x) = yy = xx + (y \oplus (\delta + \mathrm{key}))$. Thus the DP of one round of XTEA is essentiallly the DP of F' and is approximated as:

$\mathrm{xdp}^{F'}(dxx, dx \to dyy) = \mathrm{xdp}^{F}(dx \to dy) \cdot \mathrm{xdp}^{+}(dy, dxx \to dyy)$.

Attention

> The pDDT contais entries of the form $(dx,\ dy,\ \mathrm{xdp}^{F}(dx \to dy))$. However, every entry in the arrays of differentials `trail` and `diff_in` contains elements of the form: $(dx,\ dyy,\ \mathrm{xdp}^{F'}(dxx, dx \to dyy))$. Although `trail` and `dif_in` do not contain the difference dxx, the latter can be easily computed noting that $dxx = dx_{-1}$, where dx_{-1} is the input difference to F from the previous round.

For more details on the search algorithm see tea_add_threshold_search .

See also

> xtea_xor_trail_search

7.29.2.3 void xtea_xor_trail_search (uint32_t *key[4]*, uint32_t *round_key[64]*, uint32_t *round_delta[64]*)

Search for XOR differential trails in block cipher XTEA: wrapper function for tea_add_threshold_search.

Parameters

key	cryptographic key of XTEA.
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.

Algorithm Outline:

The procedure operates as follows:

1. Compute a pDDT for F (xtea_f_xor_pddt).
2. Execute the search for differential trails for n rounds (n = NROUNDS) through a successive application of xtea_xor_threshold_search :
   - Compute the best found probability on 1 round: $B[0]$.
   - Using $B[0]$ compute the best found probability on 2 rounds: $B[1]$.
   - $\ldots$
   - Using $B[0],\ldots,B[i-1]$ compute the best found probability on $(i+1)$ rounds: $B[i]$.
   - $\ldots$
   - Using $B[0],\ldots,B[n-2]$ compute the best found probability on n rounds: $B[n-1]$.
3. Print the best found trail on n rounds on standrad output and terminate.

See also

xtea_xor_threshold_search

7.30 include/xtea.hh File Reference

Header file for xtea.cc. Common functions used in the analysis of block cipher XTEA. .

Defines

- #define XTEA_XOR_P_THRES 0.120
- #define XTEA_ADD_P_THRES 0.05
- #define XTEA_XOR_MAX_PDDT_SIZE (1U $<<$ 20)
- #define XTEA_ADD_MAX_PDDT_SIZE (1U $<<$ 20)

Functions

- void xtea_r (uint32_t nrounds, uint32_t v[2], uint32_t const k[4], uint32_t lsh_-const, uint32_t rsh_const)
- uint32_t xtea_f (uint32_t x, uint32_t k, uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- uint32_t xtea_f_i (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t x_in, const uint32_t k, const uint32_t delta)
- uint32_t xtea_f2 (uint32_t xx, uint32_t x, uint32_t k, uint32_t delta, uint32_t lsh_-const, uint32_t rsh_const)

- uint32_t xtea_f2_i (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t xx_in, const uint32_t x_in, const uint32_t k, const uint32_t delta)
- uint32_t xtea_f_lxr (uint32_t x, uint32_t lsh_const, uint32_t rsh_const)
- uint32_t xtea_f_lxr_i (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t x_in)
- void xtea_all_round_keys_and_deltas (uint32_t key[4], uint32_t round_key[64], uint32_t round_delta[64])
- double **xtea_one_round_xor_differential_exper** (uint64_t npairs, int round_idx, uint32_t key, uint32_t delta, uint32_t daa, uint32_t da, uint32_t db)
- double **xtea_one_round_add_differential_exper** (uint64_t npairs, int round_idx, uint32_t key, uint32_t delta, uint32_t da, uint32_t db)
- double **xtea_xor_differential_exper_v2** (uint64_t npairs, int r, uint32_t key[4], uint32_t da[2], uint32_t db[2], uint32_t lsh_const, uint32_t rsh_const)
- double **xtea_add_differential_exper_v2** (uint64_t npairs, int r, uint32_t key[4], uint32_t da[2], uint32_t db[2], uint32_t lsh_const, uint32_t rsh_const)
- uint32_t **xtea_xor_verify_differential** (uint32_t nrounds, uint32_t npairs, uint32_t lsh_const, uint32_t rsh_const, uint32_t key[4], uint32_t dxx_init, differential_t trail[NROUNDS])
- uint32_t **xtea_add_verify_differential** (uint32_t nrounds, uint32_t npairs, uint32_t lsh_const, uint32_t rsh_const, uint32_t key[4], differential_t trail[NROUNDS])
- uint32_t **xtea_xor_verify_trail** (uint32_t nrounds, uint32_t npairs, uint32_t round_key[64], uint32_t round_delta[64], uint32_t dxx_init, differential_t trail[NROUNDS])
- uint32_t **xtea_add_verify_trail** (uint32_t nrounds, uint32_t npairs, uint32_t round_key[64], uint32_t round_delta[64], differential_t trail[NROUNDS])

7.30.1 Detailed Description

Header file for xtea.cc. Common functions used in the analysis of block cipher XTEA. .

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.30.2 Define Documentation

7.30.2.1 #define XTEA_ADD_MAX_PDDT_SIZE (1U << 20)

Maximum size of the pDDT for ADD differences.

7.30.2.2 #define XTEA_ADD_P_THRES 0.05

Probability threshold for ADD differences.

7.30.2.3 #define XTEA_XOR_MAX_PDDT_SIZE (1U << 20)

Maximum size of the pDDT for XOR differences.

7.30.2.4 #define XTEA_XOR_P_THRES 0.120

Probability threshold for XOR differences.

7.30.3 Function Documentation

7.30.3.1 void xtea_all_round_keys_and_deltas (uint32_t *key[4]*, uint32_t *round_key[64]*, uint32_t *round_delta[64]*)

Compute all round keys and round constants of block cipher XTEA.

Parameters

key	initial key.
round_key	all round keys.
round_delta	all round constants δ of XTEA.

7.30.3.2 uint32_t xtea_f (uint32_t *x*, uint32_t *k*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

The F-function of block cipher XTEA: $F(x) = ((((x \ll 4) \oplus (x \gg 5)) + x) \oplus (k + \delta)$.

Parameters

x	input to F.
k	round key.
delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$F(x)$

7.30.3.3 uint32_t xtea_f2 (uint32_t *xx*, uint32_t *x*, uint32_t *k*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

The F-function of block cipher XTEA including the modular addition with the input to the previous Fesitel round. It is denoted by F' and is defined as:

$F'(xx, x) = xx + F(x)$,

where $F(x)$ is the XTEA F-function (xtea_f).

Parameters

x	first input to F'.
xx	second input to F'.
k	round key.
delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$F(x,xx)$

7.30.3.4 uint32_t **xtea_f2_i** (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *xx_in*, const uint32_t *x_in*, const uint32_t *k*, const uint32_t *delta*)

The F'-function of block cipher XTEA (xtea_f2) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	first input to F'.
xx_in	second input to F'.
k	round key.
delta	round constant.

Returns

$F'(x,xx) \bmod 2^i$

Attention

the initial values `x_in` and `xx_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f_i()

7.30.3.5 uint32_t **xtea_f_i** (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *x_in*, const uint32_t *k*, const uint32_t *delta*)

The F-function of block cipher XTEA (xtea_f) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	input to F.
k	round key.
delta	round constant.

Returns

$F(x) \bmod 2^i$

Attention

the initial value `x_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f_lxr_i()

7.30.3.6 uint32_t xtea_f_lxr (uint32_t *x*, uint32_t *lsh_const*, uint32_t *rsh_const*)

This function represents a sub-component of the XTEA F-function denoted by f_{LXR} and defined as: $f_{\mathrm{LXR}}(x) = (((x \ll 4) \oplus (x \gg 5))$.

Note

With f_{LXR}, the F-function of XTEA (xtea_f) is expressed as: $F(x) = (f_{\mathrm{LXR}}(x) + x) \oplus (k + \delta)$.

Parameters

x	input to f_{LXR}.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$f_{\mathrm{LXR}}(x)$

7.30.3.7 uint32_t xtea_f_lxr_i (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *x_in*)

The component f_{LXR} of the XTEA F-function (xtea_f_lxr) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	first input to f_{LXR}.

Returns

$f_{\mathrm{LXR}}(x) \bmod 2^i$

Attention

the initial value `x_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f_i()

7.30.3.8 void xtea_r (uint32_t *nrounds*, uint32_t *v[2]*, uint32_t const *k[4]*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Round-reduced version of block cipher XTEA. Reference: `https://en.wikipedia.org/wiki/XTEA`.

Parameters

nrounds	number of rounds (1 ≤ `nrounds` ≤ 64).
v	plaintext.
k	secret key.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

7.31 src/adp-lsh-program.cc File Reference

The probability $\mathrm{adp}^{\ll}$ with user-provided input.

`#include "common.hh" #include "adp-shift.hh"`

Functions

- void **adp_lsh_program** ()
- int main ()

7.31.1 Detailed Description

The probability $\mathrm{adp}^{\ll}$ with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.31.2 Function Documentation

7.31.2.1 int main ( )

Main function for the ADP-LSH program.

7.32 src/adp-rsh-program.cc File Reference

The probability $\mathrm{adp}^{\gg}$ with user-provided input.

`#include "common.hh" #include "adp-shift.hh"`

Functions

- void **adp_rsh_program** ()
- int main ()

7.32.1 Detailed Description

The probability $\mathrm{adp}^{\gg}$ with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.32.2 Function Documentation

7.32.2.1 int main ( )

Main function for the ADP-RSH program.

7.33 src/adp-rsh-xor.cc File Reference

The ADD differential probability of right shift followed by XOR: $\mathrm{adp}^{\gg\oplus}$.

`#include "common.hh"` `#include "adp-xor.hh"` `#include "adp-shift.hh"` `#include "adp-rsh-xor.hh"`

Functions

- uint32_t rsh_xor (uint32_t a, uint32_t x, int r)
- double adp_rsh_xor_exper (const uint32_t da, const uint32_t dx, const uint32_t db, const int r)
- void adp_rsh_xor_alloc_matrices (gsl_matrix ∗A[3][2][2][2])
- void adp_rsh_xor_free_matrices (gsl_matrix ∗A[3][2][2][2])
- void adp_rsh_xor_normalize_matrices (gsl_matrix ∗A[3][2][2][2])
- void adp_rsh_xor_print_matrices (gsl_matrix ∗A[3][2][2][2])
- void adp_rsh_xor_sf (gsl_matrix ∗A[3][2][2][2])
- double adp_rsh_xor (gsl_matrix ∗A[3][2][2][2], uint32_t da, uint32_t dx, uint32_t db, int r)
- double adp_rsh_xor_approx (uint32_t da, uint32_t dx, uint32_t db, int r)

7.33.1 Detailed Description

The ADD differential probability of right shift followed by XOR: $\mathrm{adp}^{\gg\oplus}$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.33.2 Function Documentation

7.33.2.1 double adp_rsh_xor (gsl_matrix ∗ *A[3][2][2][2],* uint32_t *da,* uint32_t *dx,* uint32_t *db,* int *r*)

The ADD differential probability of $(\gg \oplus)$ (RSH-XOR) computed experimentally over all inputs. Complexity: $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.
da	input difference.
dx	input difference.
db	output difference.
r	shift constant.

Returns

$\text{adp}^{\gg\oplus}(r|da,dx \rightarrow db)$.

See also

adp_rsh_xor_exper

7.33.2.2 void adp_rsh_xor_alloc_matrices (gsl_matrix ∗ *A[3][2][2][2]*)

Allocate memory for the transition probability matrices for $\text{adp}^{\gg\oplus}$.

Parameters

A	transition probability matrices for $\text{adp}^{\gg\oplus}$.

See also

adp_rsh_xor_free_matrices

7.33.2.3 double adp_rsh_xor_approx (uint32_t *da*, uint32_t *dx*, uint32_t *db*, int *r*)

Approximation of $\text{adp}^{\gg\oplus}$ obtained as the multiplication of the differential probabilities $\text{adp}^{\gg}$ and $\text{adp}^{\oplus}$.

Parameters

da	input difference.
dx	input difference.
db	output difference.
r	shift constant.

Returns

$\text{adp}^{\gg\oplus}(r|da,dx \rightarrow db) \approx \text{adp}^{\gg} \cdot \text{adp}^{\oplus}$.

See also

adp_xor, adp_rsh

7.33.2.4 double adp_rsh_xor_exper (const uint32_t *da*, const uint32_t *dx*, const uint32_t *db*, const int *r*)

The ADD differential probability of RSH-XOR computed experimentally over all inputs. Complexity: $O(2^{2n})$.

Parameters

da	input difference.
dx	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\gg\oplus}(r|da, dx \rightarrow db)$.

See also

adp_rsh_xor

7.33.2.5 void adp_rsh_xor_free_matrices (gsl_matrix ∗ A[3][2][2][2])

Free memory reserved for the transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

See also

adp_rsh_xor_alloc_matrices

7.33.2.6 void adp_rsh_xor_normalize_matrices (gsl_matrix ∗ A[3][2][2][2])

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

7.33.2.7 void adp_rsh_xor_print_matrices (gsl_matrix ∗ A[3][2][2][2])

Print the elements of A.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\gg\oplus}$.

7.33.2.8 void adp_rsh_xor_sf (gsl_matrix ∗ A[3][2][2][2])

S-function for the operation $(\gg\oplus)$ (RSH-XOR).

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\mathrm{adp}^{\gg\oplus}$.

$A[3][2][2][2] = A[j][da[i]][dx[i+r]][db[i]]$, where $da[i]$ denotes the i-th bit of da, n is the word size, r is the shift constant, i is the bit position and j is a special bit position with three possible values:

- $j = 0 : 0 \leq i < n-r$.
- $j = 1 : n-r < i < n$.
- $j = 2 : i = n-r$.

7.33.2.9 uint32_t **rsh_xor** (uint32_t *a*, uint32_t *x*, int *r*)

The sequence of operations right shift (RSH) followed by an XOR (RSH-XOR).

Parameters

a	input to XOR.
x	input to RSH.
r	shift constant.

Returns

$b = a \oplus (x \gg r)$.

7.34 src/adp-shift.cc File Reference

The ADD differential probability of left shift (LSH): $\mathrm{adp}^{\ll}$ and right shift (RSH): $\mathrm{adp}^{\gg}$.

`#include "common.hh" #include "adp-shift.hh"`

Functions

- double adp_lsh_exper (uint32_t da, uint32_t db, int l)
- double adp_lsh (uint32_t da, uint32_t db, int l)
- double adp_rsh_exper (const uint32_t da, const uint32_t db, const int r)
- void adp_rsh_odiffs (uint32_t dx[4], const uint32_t da, int r)
- double adp_rsh (uint32_t da, uint32_t db, int r)

7.34.1 Detailed Description

The ADD differential probability of left shift (LSH): $\mathrm{adp}^{\ll}$ and right shift (RSH): $\mathrm{adp}^{\gg}$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.34.2 Function Documentation

7.34.2.1 double adp_lsh (uint32_t *da*, uint32_t *db*, int *l*)

The ADD differential probability of $(\lll)$ (LSH). Complexity: $O(1)$.

Parameters

da	input difference.
db	output difference.
l	shift constant.

Returns

$\mathrm{adp}^{\lll}(l|\ da \rightarrow db)$.

See also

adp_lsh_exper

7.34.2.2 double adp_lsh_exper (uint32_t *da*, uint32_t *db*, int *l*)

The ADD differential probability of $(\lll)$ (LSH) computed experimentally over all inputs. Complexity: $O(2^n)$.

Parameters

da	input difference.
db	output difference.
l	shift constant.

Returns

$\mathrm{adp}^{\lll}(l|\ da \rightarrow db)$.

See also

adp_lsh

7.34.2.3 double adp_rsh (uint32_t *da*, uint32_t *db*, int *r*)

The ADD differential probability of $(\gg)$ (RSH). Complexity: $O(1)$.

Parameters

da	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\gg}(r|\ da \to db)$.

See also

adp_rsh_exper

Note

$db \in \{(da \gg 5), (da \gg 5) + 1, (da \gg 5) - 2^{n-5}, (da \gg 5) - 2^{n-5} + 1\}$.

7.34.2.4 double adp_rsh_exper (const uint32_t *da*, const uint32_t *db*, const int *r*)

The ADD differential probability of $(\gg)$ (RSH) computed experimentally over all inputs. Complexity: $O(2^n)$.

Parameters

da	input difference.
db	output difference.
r	shift constant.

Returns

$\mathrm{adp}^{\gg}(r|\ da \to db)$.

See also

adp_rsh

7.34.2.5 void adp_rsh_odiffs (uint32_t *dx[4]*, const uint32_t *da*, int *r*)

Compute the set of possible output differences dx after a right shift by r

Parameters

da	input difference.
r	shift constant.
dx	the set of all 4 possible output differences.

7.35 src/adp-tea-f-fk-ddt.cc File Reference

Computing the full difference distribution table (DDT) for the F-function of block cipher TEA by exaustive search over all inputs. Complexity $O(2^{2n})$.

`#include "common.hh" #include "tea.hh"`

Functions

- void ddt_sort_rows (differential_t **T)
- bool comp_rows (differential_t *a, differential_t *b)
- void ddt_sort_first_col (differential_t **T)
- void ddt_to_list (uint32_t **DDT, differential_t *SDDT)
- void ddt_to_diff_struct (uint32_t **DDT, differential_t **SDDT)
- void ddt_sort (differential_t *SDDT)
- void print_rsddt (differential_t **RSDDT)
- void print_sddt (differential_t *SDDT)
- double adp_f_exper_fixed_key_all (const uint32_t da, const uint32_t db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_adp_f_exper_fixed_key_all (const uint32_t da, uint32_t *db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- differential_t ** rsddt_alloc ()
- void rsddt_free (differential_t **T)
- differential_t * sddt_alloc ()
- void sddt_free (differential_t *ST)
- uint32_t ** ddt_alloc ()
- void ddt_free (uint32_t **T)
- void ddt_f (uint32_t **T, uint32_t k0, uint32_t k1, uint32_t delta, uint32_t lsh_-const, uint32_t rsh_const)
- void ddt_print (uint32_t **T)
- double adp_f_ddt (uint32_t **DDT, uint32_t dx, uint32_t dy)
- double max_adp_f_ddt (uint32_t **DDT, uint32_t dx, uint32_t *dy)
- double max_adp_f_rsddt (differential_t **TS, uint32_t dx, uint32_t *dy)
- uint32_t *** xddt_alloc ()
- void xddt_free (uint32_t ***T)
- differential_t *** xrsddt_alloc ()
- void xrsddt_free (differential_t ***T)
- differential_t ** xsddt_alloc ()
- void xsddt_free (differential_t **ST)

7.35.1 Detailed Description

Computing the full difference distribution table (DDT) for the F-function of block cipher TEA by exaustive search over all inputs. Complexity $O(2^{2n})$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013 All functions in this file have exponential complexity in the word size. They are useful only for verifying other computations on small word sizes, typically $n \leq 10$.

7.35.2 Function Documentation

7.35.2.1 double adp_f_ddt (uint32_t ** *DDT*, uint32_t *dx*, uint32_t *dy*)

Compute the ADD differential probability of the TEA F-function from the full DDT, pre-computed for a a fixed key and round constant.

Parameters

DDT	DDT.
dx	input difference.
dy	output difference.

Returns

DDT[da][db] = $\mathrm{adp}^F(k_0, k_1, \delta | \ da \rightarrow dd)$.

See also

adp_f_exper_fixed_key_all

7.35.2.2 double adp_f_exper_fixed_key_all (const uint32_t *da*, const uint32_t *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the ADD differential probability of the TEA F-function for a fixed key and round constants ($\mathrm{adp}^F(k_0, k_1, \delta | \ da \rightarrow dd)$) by exhaustive searc over all inputs. Complexity $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta | \, da \rightarrow dd)$.

7.35.2.3 bool comp_rows (differential_t * *a*, differential_t * *b*)

Compare two rows in a row-sorted DDT by their first (max) element. Assumes that the elemnets in a row are sorted in descending order.

Parameters

a	row of differentials in a DDT.
b	row of differentials in a DDT.

7.35.2.4 uint32_t** ddt_alloc ( )

Allocate memory for a DDT as a 2D array containingg number of rigt pairs.

Returns

a DDT as a 2D array containing number of rigt pairs.

See also

ddt_free

7.35.2.5 void ddt_f (uint32_t ** *T*, uint32_t *k0*, uint32_t *k1*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the full difference distribution table (DDT) for the F-function of block cipher TEA for a fixed key and round constant, by exaustive search over all input values and differences. Complexity $O(2^{2n})$.

Parameters

T	a DDT as a 2D array containing number of right pairs..
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

full DDT for the TEA F-function.

7.35.2.6 void ddt_free (uint32_t ** *T*)

Free the memory reserved for a DDT as a 2D array containingg number of rigt pairs.

Parameters

T	a DDT as a 2D array containing number of rigt pairs.

See also

ddt_alloc

7.35.2.7 void ddt_print (uint32_t ** *T*)

Print the entries of a DDT.

Parameters

T	DDT.

7.35.2.8 void ddt_sort (differential_t * *SDDT*)

Sort all elements of a DDT, represented as a 1D list of differentials, in descending order by the number of right pairs.

Parameters

SDDT	DDT as a 1D list of differentials.

7.35.2.9 void ddt_sort_first_col (differential_t ** *T*)

Sorts the rows of a difference distribution table (DDT) 2D by the probability of the elements in the first column -- highest probability first.

Parameters

T	a difference distribution table (DDT).

7.35.2.10 void ddt_sort_rows (differential_t ** *T*)

Sort every row by decreasing number of right pairs.

Parameters

T	a difference distribution table (DDT).

7.35.2.11 void ddt_to_diff_struct (uint32_t ** *DDT,* differential_t ** *SDDT*)

Convert a DDT to 2D array of differentials.

Parameters

DDT	difference distribution table.
SDDT	array differentials.

7.35.2.12 void ddt_to_list (uint32_t ** *DDT*, differential_t * *SDDT*)

Convert a DDT to a list of differentials.

Parameters

DDT	difference distribution table.
SDDT	list of differentials.

7.35.2.13 double max_adp_f_ddt (uint32_t ** *DDT*, uint32_t *dx*, uint32_t * *dy*)

For a fixed input difference to the TEA F-function compute the maximum probability output ADD difference from the full DDT, precomputed for a fixed key and round constant. Complexity $O(1)$.

Parameters

DDT	DDT.
dx	input difference.
dy	maximum probability output difference.

Returns

$\max_{dd} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

max_adp_f_exper_fixed_key_all

7.35.2.14 double max_adp_f_exper_fixed_key_all (const uint32_t *da*, uint32_t * *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For a fixed input difference to the TEA F-function compute the maximum probability output ADD difference for a fixed key and round constant by exhaustive search over all inputs and output differences. Complexity $O(2^{2n})$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dd} \mathrm{adp}^F(k_0, k_1, \delta | \; da \rightarrow dd)$.

See also

max_adp_f_ddt, max_adp_f_rsddt

7.35.2.15 double max_adp_f_rsddt (differential_t ** *TS*, uint32_t *dx*, uint32_t * *dy*)

For a fixed input difference to the TEA F-function compute the maximum probability output ADD difference from the full DDT represented as a 2D array of differentials. In this DDT the differentials in every row are sorted by decreasing probability. Complexity $O(1)$.

Parameters

TS	a DDT in which the differentials in every row are sorted by decreasing number of probability.
dx	input difference.
dy	maximum probability output difference.

Returns

TS[dx][0] = $\max_{dd} \mathrm{adp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$.

See also

max_adp_f_ddt, max_adp_f_exper_fixed_key_all

7.35.2.16 void print_rsddt (differential_t ** *RSDDT*)

Print the elements of a DDT.

Parameters

RSDDT	DDT as a 2D list of differentials.

7.35.2.17 void print_sddt (differential_t * *SDDT*)

Print the elements of a DDT.

Parameters

SDDT	DDT as a 1D list of differentials.

7.35.2.18 differential_t** rsddt_alloc ( )

Allocate memory for a DDT as a 2D array of differentials.

Returns

a DDT as a 2D array of differentials.

See also

rsddt_free

7.35.2.19 void rsddt_free (differential_t ** *T*)

Free the memory reserved for a DDT as a 2D array of differentials.

Parameters

T	a DDT as a 2D array of differentials.

See also

rsddt_alloc

7.35.2.20 differential_t* sddt_alloc ( )

Allocate memory for a DDT as a 1D array of differentials.

Returns

a DDT as a 1D array of differentials.

See also

sddt_free

7.35.2.21 void sddt_free (differential_t * *ST*)

Free the memory reserved for a DDT as a 1D array of differentials.

Parameters

ST	a DDT as a 1D array of differentials.

See also

sddt_alloc

7.35.2.22 uint32_t*** xddt_alloc ( )

Allocate memory for a an array of NDELTA DDTs. Each DDT represents a 2D array containing numbers of rigt right pairs and generated for a fixed value of the δ constant of the TEA F-function.

Returns

array of DDTs: each DDT represents a 2D array containing number of right pairs.

7.35.2.23 void xddt_free (uint32_t $***$ *T*)

Free the memory reserved from a previous call to xddt_alloc()

Parameters

T	an array of DDTs: each DDT is a 2D array containing number of right pairs.

7.35.2.24 differential_t$***$ xrsddt_alloc ( )

Allocate memory for a an array of NDELTA DDTs. Each DDT represents a 2D array of differentials, generated for a fixed value of the δ constant of the TEA F-function.

Returns

array of DDTs: each DDT represents a 2D array of differentials.

7.35.2.25 void xrsddt_free (differential_t $***$ *T*)

Free the memory reserved from a previous call to xrsddt_alloc()

Parameters

T	an array of DDTs: each DDT is a 2D array of differentials.

7.35.2.26 differential_t$**$ xsddt_alloc ( )

Allocate memory for a an array of NDELTA DDTs. Each DDT represents a 1D list of differentials, generated for a fixed value of the δ constant of the TEA F-function.

Returns

array of DDTs: each DDT represents a 1D list of differentials,

7.35.2.27 void xsddt_free (differential_t $**$ *ST*)

Free the memory reserved from a previous call to xrsddt_free()

Parameters

ST	an array of DDTs: each DDT is a 1D list of differentials.

7.36 src/adp-tea-f-fk-noshift.cc File Reference

The additive differential probability (ADP) of a modified version of the F-function of TEA with the shift operations removed. Complexity $O(n)$.

```
#include "common.hh" #include "tea.hh" #include "adp-tea-f-fk-noshift.-
hh"
```

Functions

- void adp_f_op_noshift_sf (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_alloc_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_free_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_normalize_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- void adp_f_op_noshift_print_matrices (gsl_matrix ∗A[NSPOS][2][2][2][2][2])
- double adp_f_op_noshift (gsl_matrix ∗A[NSPOS][2][2][2][2][2], uint32_t k0, uint32_t k1, uint32_t delta, uint32_t da, uint32_t db)
- double adp_f_op_noshift_exper (uint32_t k0, uint32_t k1, uint32_t delta, uint32_t da, uint32_t db)

7.36.1 Detailed Description

The additive differential probability (ADP) of a modified version of the F-function of T-EA with the shift operations removed. Complexity $O(n)$. The F-function of TEA with the shift operations removed is denoted by F' and is defined as: $y = F'(k_0, k_1, \delta | x) = (x+k_0) \oplus (x+\delta) \oplus (x+k_1)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.36.2 Function Documentation

7.36.2.1 double adp_f_op_noshift (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2],* uint32_t *k0,* uint32_t *k1,* uint32_t *delta,* uint32_t *da,* uint32_t *db*)

The additive differential probability (ADP) of a modified version of the F-function of TEA with the shift operations removed, denoted by F' and defined as:

$y = F'(k_0, k_1, \delta | x) = (x+k_0) \oplus (x+\delta) \oplus (x+k_1)$.

Complexity: $O(n)$.

Parameters

A	transition probability matrices for F' computed with adp_f_op_noshift_-sf
k0	first round key.
k1	second round key.
delta	round constant.
da	input difference.
db	output difference.

Returns

$\mathrm{adp}^{F'}(k_0, k_1, \delta | \, da \rightarrow db)$.

7.36.2.2 void adp_f_op_noshift_alloc_matrices (gsl_matrix ∗ *A[NSPOS][2][2][2][2][2]*)

Allocate memory for the transition probability matrices for F'.

Parameters

A	transition probability matrices for F'.

See also

adp_rsh_xor_free_matrices

7.36.2.3 double adp_f_op_noshift_exper (uint32_t *k0,* uint32_t *k1,* uint32_t *delta,* uint32_t *da,* uint32_t *db*)

The additive differential probability (ADP) of F' (a modified version of the F-function of TEA with the shift operations removed) computed experimentally over all inputs. - Complexity: $O(2^{2n})$.

Parameters

k0	first round key.
k1	second round key.
delta	round constant.
da	input difference.
db	output difference.

Returns

$\mathrm{adp}^{F'}(k_0, k_1, \delta | \, da \rightarrow db)$.

See also

adp_f_op_noshift

7.36.2.4 void adp_f_op_noshift_free_matrices (gsl_matrix ∗ A[NSPOS][2][2][2][2][2])

Free memory reserved by a previous call to adp_rsh_xor_free_matrices.

Parameters

A	transition probability matrices for F'.

7.36.2.5 void adp_f_op_noshift_normalize_matrices (gsl_matrix ∗ A[NSPOS][2][2][2][2][2])

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for F'.

7.36.2.6 void adp_f_op_noshift_print_matrices (gsl_matrix ∗ A[NSPOS][2][2][2][2][2])

Print the elements of A.

Parameters

A	transition probability matrices for F'.

7.36.2.7 void adp_f_op_noshift_sf (gsl_matrix ∗ A[NSPOS][2][2][2][2][2])

S-function for a modified version of the TEA F-function with the shift operations removed, denoted by F' and defined as:

$y = F'(k_0, k_1, \delta | \, x) = (x + k_0) \oplus (x + \delta) \oplus (x + k_1)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for F'.

$A[j][2][2][2][2][2] = A[j][k0[i]][k1[i]][\delta[i]][da[i]][db[i]]$, where

- j : dummy variable for future use.
- $k_0[i]$: the i-th bit of the first round key.
- $k_1[i]$: the i-th bit of the second round key.
- $\delta[i]$: the i-th bit of the round constant:
- $da[i]$: the i-th bit of the input difference.
- $db[i]$: the i-th bit of the output difference.

7.37 src/adp-tea-f-fk.cc File Reference

The ADD differential probability of the F-function of TEA for a fixed key and round constants $\mathrm{adp}^F(k_0, k_1, \delta | \; da \to dd)$, where $F(k_0, k_1, \delta | \; x) = ((x \ll 4) + k_0) \oplus (x + \delta) \oplus ((x \gg 5) + k_1)$ Complexity: $O(n) < c \leq O(2^n)$.

`#include "common.hh" #include "tea.hh"`

Functions

- bool adp_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool adp_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, int32_t x)
- uint32_t adp_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double adp_f_fk (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint32_t adp_f_assign_bit_x_dx (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dx)
- double max_dx_adp_f_fk (const uint32_t n, uint32_t ∗ret_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint32_t adp_f_assign_bit_x_dy (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dy)
- double max_dy_adp_f_fk (const uint32_t n, const uint32_t dx, uint32_t ∗ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double all_dy_adp_f_fk (const uint32_t n, const uint32_t dx, uint32_t ∗ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, uint64_t ∗x_cnt)
- uint32_t adp_f_assign_bit_x_dx_dy (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, differential_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dx, uint32_t ∗ret_dy)
- double max_dx_dy_adp_f_fk (const uint32_t n, uint32_t ∗ret_dx, uint32_t ∗ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint64_t ∗∗∗ x_cnt_alloc ()

- void x_cnt_free (uint64_t $***$x_cnt)
- void x_cnt_print (uint32_t $***$x_cnt)
- uint32_t adp_f_assign_bit_x_dx_key (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh-_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32-_t dx, const uint32_t dy, uint64_t $***$x_cnt, double $*$ret_prob, uint32_t $*$ret_dx, uint32_t $*$ret_k0, uint32_t $*$ret_k1)
- double max_key_dx_adp_f_fk (const uint32_t n, uint32_t $*$ret_dx, const uint32_t dy, uint32_t $*$ret_k0, uint32_t $*$ret_k1, const uint32_t delta, const uint32_t lsh_-const, const uint32_t rsh_const)
- double adp_f_fk_v2 (const uint32_t da, const uint32_t dd, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_-const)
- void f_sfun (const uint32_t n, const uint32_t x_word, const uint32_t dx_word, const uint32_t delta_word, const uint32_t k0_word, const uint32_t k1_word)
- double adp_f_fk_exper (const uint32_t da, const uint32_t db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_dx_adp_f_fk_exper (uint32_t $*$max_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32-_t rsh_const)
- double max_dy_adp_f_fk_exper (const uint32_t dx, uint32_t $*$max_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32-_t rsh_const)
- double max_dx_dy_adp_f_fk_exper (uint32_t $*$max_dx, uint32_t $*$max_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)

7.37.1 Detailed Description

The ADD differential probability of the F-function of TEA for a fixed key and round constants $\mathrm{adp}^F(k_0,k_1,\delta|\ da \rightarrow dd)$, where $F(k_0,k_1,\delta|\ x) = ((x \ll 4)+k_0) \oplus (x+\delta) \oplus ((x \gg 5)+k_1)$ Complexity: $O(n) < c \leq O(2^n)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Attention

The algorithms in this file have complexity that depends on the input and output differences to F. It is worst-case exponential in the word size, but is sub-exponential on average.

7.37.2 Function Documentation

7.37.2.1 uint32_t **adp_f_assign_bit_x** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the F-function of TEA is satisfied. The function operates by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i + rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key ADD probability of `F`: $\mathrm{adp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_fk

7.37.2.2 uint32_t **adp_f_assign_bit_x_dx** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dx*)

For given output difference `dy`, compute all input differences `dx` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability input difference.

The function works by recursively assigning the bits of `x` and `dx` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat .

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dx`.
ret_prob	the maximum probability over all input differences $\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dx	the input difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_assign_bit_x, max_dx_adp_f_fk

7.37.2.3 uint32_t **adp_f_assign_bit_x_dx_dy** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, **differential_t** ∗ *x_cnt*, double ∗ *ret_prob*, uint32_t ∗ *ret_dx*, uint32_t ∗ *ret_dy*)

For the TEA F-functuion with fixed key and round constant, compute all differentials $(dx \rightarrow dy)$ and their probabilities.

The function works by recursively assigning the bits of `x`, `dx` and `dy` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.

delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^{2n} differentials $(dx \rightarrow dy)$ and their probabilities.
ret_prob	the maximum probability over all input and output differences $\max_{dy,dy} \mathrm{adp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dx	the input difference of the maximum probability differential.
ret_dy	the output difference of the maximum probability differential.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_assign_bit_x_dx, adp_f_assign_bit_x_dy.

7.37.2.4 uint32_t **adp_f_assign_bit_x_dx_key** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t $***$ *x_cnt*, double $*$ *ret_prob*, uint32_t $*$ *ret_dx*, uint32_t $*$ *ret_k0*, uint32_t $*$ *ret_k1*)

For the TEA F-function with fixed round constant, and for a fixed output difference `dy`, compute all differentials $(dx \rightarrow dy)$ and their probabilities for all values of the round keys `k0`, `k1`.

The function works by recursively assigning the bits of `x`, `dx`, `k0` and `k1` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^{3n} differentials $(dx \rightarrow dy)$ and their probabilities.
ret_prob	the maximum probability over all input differences and round keys $\max_{dx,k_0,k_1} \mathrm{adp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dx	the input difference of the maximum probability differential.
ret_k0	the first round key for the maximum probability differential.
ret_k1	the second round key for the maximum probability differential.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_f_assign_bit_x_dx_key, adp_f_assign_bit_x_dx_dy.

7.37.2.5 uint32_t **adp_f_assign_bit_x_dy** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dy*)

For given input difference `dx`, compute all output differences `dy` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability output difference.

The function works by recursively assigning the bits of `x` and `dy` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dy`.
ret_prob	the maximum probability over all output differences $\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta \| \, dx \rightarrow dy)$.
ret_dy	the output difference that has maximum probability.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_f_assign_bit_x_dx

7.37.2.6 bool adp_f_check_x (const uint32_t *lsh_const,* const uint32_t *rsh_const,* const uint32_t *k0,* const uint32_t *k1,* const uint32_t *delta,* const uint32_t *dx,* const uint32_t *dy,* const uint32_t *x*)

Check if a given value x satisfies the ADD differential $(dx \rightarrow dy)$ for the TEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k_0, k_1, \delta: \ dy = F(x + dx) - F(x)$.

7.37.2.7 double adp_f_fk (const uint32_t *n,* const uint32_t *dx,* const uint32_t *dy,* const uint32_t *k0,* const uint32_t *k1,* const uint32_t *delta,* const uint32_t *lsh_const,* const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher TEA: $\mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$.

See also

adp_f_assign_bit_x

7.37.2.8 double adp_f_fk_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher TEA: $\mathrm{adp}^F(k_0, k_1, \delta | \ dx \to dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta | \ dx \to dy)$.

See also

adp_f_fk, adp_f_fk_v2.

7.37.2.9 double adp_f_fk_v2 (const uint32_t *da*, const uint32_t *dd*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher TEA: $\mathrm{adp}^F(k_0, k_1, \delta | \ dx \to dy)$.

The function works by dividing the input to F into independent parts and iterating over the values in each part. The resulting complexity is equivalent to exhaustive search over all inputs: $O(2^n)$.

Parameters

da	input difference.
dd	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k_0, k_1, \delta | \ dx \to dy)$.

See also

adp_f_fk

7.37.2.10 bool adp_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, int32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` is satisfied on the `i` LS bits of `x` i.e. check if $k_0, k_1, \delta : \; dy[i-1:0] = F(x[i-1:0] + dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i + rsh_const`).

Returns

TRUE if $k_0, k_1, \delta : \; dy[i-1:0] = F(x[i-1:0] + dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

7.37.2.11 double all_dy_adp_f_fk (const uint32_t *n*, const uint32_t *dx*, uint32_t * *ret_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, uint64_t * *x_cnt*)

For given input difference `dx`, compute all output differences `dy` for the TEA F-function with fixed keys and round constants. Returns the maximum output probability.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
x_cnt	array of 2^n counters - each one keeps track of the number of inputs `x` satisfying $(dx \rightarrow dy)$ for every `dy`.

Returns

$\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$.

See also

max_dy_adp_f_fk

7.37.2.12 void f_sfun (const uint32_t *n*, const uint32_t *x_word*, const uint32_t *dx_word*, const uint32_t *delta_word*, const uint32_t *k0_word*, const uint32_t *k1_word*)

Compute the S-function for the TEA F function.

Parameters

n	word size.
x_word	input to F.
dx_word	input difference.
delta_word	round constant.
k0_word	first round key.
k1_word	second round key.

7.37.2.13 double max_dx_adp_f_fk (const uint32_t *n*, uint32_t * *ret_dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
ret_dx	maximum probability input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dx

7.37.2.14 double max_dx_adp_f_fk_exper (uint32_t * max_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

max_dx	maximum probability input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$.

See also

max_dx_adp_f_fk

7.37.2.15 double max_dx_dy_adp_f_fk (const uint32_t n, uint32_t * ret_dx, uint32_t * ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)

For the TEA F-functuion with fixed key and round constant, compute the maximum probability differential $(dx \rightarrow dy)$ over all input and output differences. **Complexity:** $O(3n) < c \leq O(2^{3n})$. **Memory:** $12 \cdot 2^{2n}$ Bytes.

Parameters

n	word size.
ret_dx	the input difference of the maximum probability differential.
ret_dy	the output difference of the maximum probability differential.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx,dy} \mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dx_dy

7.37.2.16 double max_dx_dy_adp_f_fk_exper (uint32_t ∗ *max_dx*, uint32_t ∗ *max_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For the TEA F-functuion with fixed key and round constant, compute the maximum probability differential $(dx \rightarrow dy)$ over all input and output differences. **Complexity:** $O(2^{3n})$.

Parameters

max_dx	the input difference of the maximum probability differential.
max_dy	the output difference of the maximum probability differential.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx,dy} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

max_dx_dy_adp_f_fk

7.37.2.17 double max_dy_adp_f_fk (const uint32_t *n*, const uint32_t *dx*, uint32_t ∗ *ret_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory requirement:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dy, max_dy_adp_f_fk

7.37.2.18 double **max_dy_adp_f_fk_exper** (const uint32_t *dx*, uint32_t ∗ *max_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx	input difference.
max_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k_0, k_1, \delta | \ dx \rightarrow dy)$.

See also

max_dy_adp_f_fk

7.37.2.19 double **max_key_dx_adp_f_fk** (const uint32_t *n*, uint32_t ∗ *ret_dx*, const uint32_t *dy*, uint32_t ∗ *ret_k0*, uint32_t ∗ *ret_k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For the TEA F-functuion with fixed key and round constant, compute the maximum probability differential $(dx \rightarrow dy)$ over all input differences and round keys. **Complexity:** $O(4n) < c \leq O(2^{4n})$. **Memory:** $12 \cdot 2^{3n}$ Bytes.

Parameters

n	word size.
dy	output difference.
ret_dx	the input difference of the maximum probability differential.
ret_k0	the first round key for the maximum probability differential.
ret_k1	the second round key for the maximum probability differential.

delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx,k_0,k_1} \mathrm{adp}^F(k_0,k_1,\delta|\ dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dx_key

7.37.2.20 uint64_t∗∗∗ x_cnt_alloc ( )

Allocate memory for a 2D array of differentials.

Returns

2D array of differentials.

See also

x_cnt_free

7.37.2.21 void x_cnt_free (uint64_t ∗∗∗ *x_cnt*)

Free the memory allocated from a previous call to x_cnt_alloc

Parameters

x_cnt	2D array of differentials.

See also

x_cnt_alloc

7.37.2.22 void x_cnt_print (uint32_t ∗∗∗ *x_cnt*)

Print the elements of a 2D array of differentials.

Parameters

x_cnt	2D array of differentials.

7.38 src/adp-xor-fi-program.cc File Reference

The probability $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ with user-provided input.

`#include "common.hh" #include "adp-xor-fi.hh"`

Functions

- void adp_xor_fixed_input_program ()
- int main ()

7.38.1 Detailed Description

The probability $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.38.2 Function Documentation

7.38.2.1 void adp_xor_fixed_input_program ( )

Compute ADP-XOR-FI with user-provided input.

7.38.2.2 int main ( )

Main function for the ADP-XOR-FI program.

7.39 src/adp-xor-fi.cc File Reference

The ADD differential probability of XOR with one fixed input (FI): $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow db)$.

`#include "common.hh" #include "adp-xor-fi.hh"`

Functions

- void adp_xor_fixed_input_alloc_matrices (gsl_matrix ∗A[2][2][2])
- void adp_xor_fixed_input_free_matrices (gsl_matrix ∗A[2][2][2])
- void adp_xor_fixed_input_normalize_matrices (gsl_matrix ∗A[2][2][2])
- void adp_xor_fixed_input_sf (gsl_matrix ∗A[2][2][2])
- double adp_xor_fixed_input (gsl_matrix ∗A[2][2][2], uint32_t a, uint32_t db, uint32_t dc)
- double adp_xor_fixed_input_exper (const uint32_t a, const uint32_t db, const uint32_t dc)

7.39.1 Detailed Description

The ADD differential probability of XOR with one fixed input (FI): $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow db)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

See also

adp-xor.cc

7.39.2 Function Documentation

7.39.2.1 double adp_xor_fixed_input (gsl_matrix ∗ *A[2][2][2]*, uint32_t *a*, uint32_t *db*, uint32_t *dc*)

The additive differential probability (ADP) of $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$. **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ computed with adp_xor_fixed_input_sf.
a	input value.
db	input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(a, db \rightarrow db)$.

7.39.2.2 void adp_xor_fixed_input_alloc_matrices (gsl_matrix ∗ *A[2][2][2]*)

Allocate memory for the transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

See also

adp_xor_fixed_input_free_matrices

7.39.2.3 double adp_xor_fixed_input_exper (const uint32_t *a*, const uint32_t *db*, const uint32_t *dc*)

The additive differential probability (ADP) of $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ computed experimentally over all inputs. Complexity: $O(2^n)$.

Parameters

a	input value.
db	input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(a, db \rightarrow db)$.

See also

adp_xor_fixed_input

7.39.2.4 void adp_xor_fixed_input_free_matrices (gsl_matrix ∗ *A[2][2][2]*)

Free memory reserved by a previous call to adp_xor_fixed_input_alloc_matrices.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

7.39.2.5 void adp_xor_fixed_input_normalize_matrices (gsl_matrix ∗ *A[2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

7.39.2.6 void adp_xor_fixed_input_sf (gsl_matrix ∗ *A[2][2][2]*)

S-function for $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$: $\mathrm{adp}^{\oplus}(a, db \rightarrow db)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A.

A[2][2][2] = A[a[i]][db[i]][dc[i]], where

- a[i] : the i-th bit of the fixed input.
- db[i] : the i-th bit of the input difference.
- dc[i] : the i-th bit of the output difference.

7.40 src/adp-xor-pddt.cc File Reference

Compute a partial difference distribution table (pDDT) for $\mathrm{adp}^{\oplus}$.

`#include "common.hh" #include "adp-xor.hh"`

Functions

- uint32_t adp_xor_ddt_exper (std::multiset< differential_3d_t, struct_comp_diff_3d_p > *diff_set, double p_thres)
- void adp_xor_pddt_i (const uint32_t k, const uint32_t n, const double p_thres, gsl_matrix *A[2][2][2], gsl_vector *C, uint32_t *da, uint32_t *db, uint32_t *dc, double *p, std::multiset< differential_3d_t, struct_comp_diff_3d_p > *diff_set)
- void adp_xor_ddt (uint32_t n, double p_thres)

7.40.1 Detailed Description

Compute a partial difference distribution table (pDDT) for $\mathrm{adp}^{\oplus}$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.40.2 Function Documentation

7.40.2.1 void adp_xor_ddt (uint32_t *n*, double *p_thres*)

Compute a partial DDT for $\mathrm{adp}^{\oplus}$: wrapper function of adp_xor_pddt_i.

Parameters

n	word size.
p_thres	probability threshold.

See also

adp_xor_pddt_i.

7.40.2.2 uint32_t adp_xor_ddt_exper (std::multiset< differential_3d_t, struct_comp_diff_3d_p > ∗ *diff_set*, double *p_thres*)

Compute a partial DDT for $\mathrm{adp}^{\oplus}$ by exhasutive search over all input and output differences.

Parameters

diff_set	set of all differentials with probability not less than the threshold (the pDDT)
p_thres	probability threshold.

Returns

number of elements in the pDDT.

See also

adp_xor_pddt_i

7.40.2.3 void adp_xor_pddt_i (const uint32_t *k*, const uint32_t *n*, const double *p_thres*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, double ∗ *p*, std::multiset< differential_3d_t, struct_comp_diff_3d_p > ∗ *diff_set*)

Recursively compute all ADD differentials $(da, db \rightarrow dc)$ for XOR that have probability $\mathrm{adp}^{\oplus}$ larger than a fixed probability threshold `p_thres`.

The function works recursively starting from the LS bit `k` = 0 and terminating at the -MS bit `n`. At every bit position i it assigns values to the i-th bits of the differences `da`, `db`, `dc` and evaluates the probability of the resulting partial (i+1)-bit differential: $(da[i:0], db[i:0] \rightarrow dc[i:0])$. The recursion proceeds only if this probability is not less than the threshold `p_thres`. When i = n, the differential $(da[n-1:0], db[n-1:0] \rightarrow dc[n-1:0])$ is stored in an STL multiset structure (internally implemented as a Red-Black tree).

The **complexity** is strongly dependent on the threshold and is worst-case exponential in the word size: $O(2^{3n})$.

Note

If `p_thres` = 0.0 then the full DDT is computed.
Can be used also to compute all differentials that have non-zero probability by setting p_thres $>$ 0.0 .
For 32 bit words, recommended values for the threshold are p_thres $>=$ 0.5.

Parameters

k	current bit position in the recursion.
n	word size.
p_thres	probability threshold.
A	transition probability matrices for $\mathrm{adp}^{\oplus}$.
C	unit column vector for computing $\mathrm{adp}^{\oplus}$ (adp_xor).
da	first input difference.
db	second input difference.
dc	output difference.
p	probability of the differential $(da[k:0], db[k:0] \rightarrow dc[k:0])$.
diff_set	set of all differentials with probability not less than the threshold (the pDDT)

7.41 src/adp-xor-program.cc File Reference

The probability $\mathrm{adp}^{\oplus}$ with user-provided input.

`#include "common.hh"` `#include "adp-xor.hh"`

Functions

- void adp_xor_program ()
- int main ()

7.41.1 Detailed Description

The probability $\mathrm{adp}^{\oplus}$ with user-provided input.

Author

V.Velichkov, `vesselin.velichkov@uni.lu`

Date

2012-2013

7.41.2 Function Documentation

7.41.2.1 void adp_xor_program ( )

Compute ADP-XOR with user-provided input.

7.41.2.2 int main ( )

Main function for the ADP-XOR program.

7.42 src/adp-xor.cc File Reference

The ADD differential probability of XOR $\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.

#include "common.hh" #include "adp-xor.hh"

Functions

- void adp_xor_alloc_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_free_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_normalize_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_print_matrices (gsl_matrix *A[2][2][2])
- void adp_xor_sf (gsl_matrix *A[2][2][2])
- double adp_xor (gsl_matrix *A[2][2][2], uint32_t da, uint32_t db, uint32_t dc)
- double adp_xor_exper (const uint32_t da, const uint32_t db, const uint32_t dc)

7.42.1 Detailed Description

The ADD differential probability of XOR $\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.42.2 Function Documentation

7.42.2.1 double adp_xor (gsl_matrix * *A[2][2][2],* uint32_t *da,* uint32_t *db,* uint32_t *dc*)

The additive differential probability of XOR ($\mathrm{adp}^{\oplus}$). **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$ computed with adp_xor_sf.
da	first input difference.
db	second input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.

See also

xdp_add

7.42.2.2 void adp_xor_alloc_matrices (gsl_matrix * A[2][2][2])

Allocate memory for the transition probability matrices for $\mathrm{adp}^{\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

See also

adp_xor_free_matrices

7.42.2.3 double adp_xor_exper (const uint32_t da, const uint32_t db, const uint32_t dc)

The additive differential probability of XOR ($\mathrm{adp}^{\oplus}$) computed experimentally over all inputs. **Complexity:** $O(2^{2n})$.

Parameters

da	first input difference.
db	second input difference.
dc	output difference.

Returns

$\mathrm{adp}^{\oplus}(da, db \rightarrow db)$.

See also

adp_xor

7.42.2.4 void adp_xor_free_matrices (gsl_matrix * A[2][2][2])

Free memory reserved by a previous call to adp_xor_alloc_matrices.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

7.42.2.5 void adp_xor_normalize_matrices (gsl_matrix * A[2][2][2])

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

7.42.2.6 void adp_xor_print_matrices (gsl_matrix ∗ *A[2][2][2]*)

Print the matrices for $\mathrm{adp}^{\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$.

7.42.2.7 void adp_xor_sf (gsl_matrix ∗ *A[2][2][2]*)

S-function for $\mathrm{adp}^{\oplus}$: $\mathrm{adp}^{\oplus}(da, db \to db)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\mathrm{adp}^{\oplus}(da, db \to db)$.

$A[2][2][2] = A[da[i]][db[i]][dc[i]]$, where

- $da[i]$: the i-th bit of the first input difference.
- $db[i]$: the i-th bit of the second input difference.
- $dc[i]$: the i-th bit of the output difference.

See also

xdp_add_sf

7.43 src/adp-xor3-program.cc File Reference

The probability ($\mathrm{adp}^{3\oplus}$) with user-provided input.

`#include "common.hh" #include "adp-xor3.hh"`

Functions

- void **adp_xor3_program** ()
- int main ()

7.43.1 Detailed Description

The probability ($\mathrm{adp}^{3\oplus}$) with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.43.2 Function Documentation

7.43.2.1 int main ( )

Main function for the ADP-XOR3 program.

7.44 src/adp-xor3.cc File Reference

The ADD differential probability of XOR with three inputs ($3\oplus$): $\mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$.

`#include "common.hh" #include "adp-xor3.hh"`

Functions

- void adp_xor3_alloc_matrices (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_free_matrices (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_print_matrices (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_print_matrices_sage (gsl_matrix ∗A[2][2][2][2])
- void adp_xor3_normalize_matrices (gsl_matrix ∗A[2][2][2][2])
- int adp_xor3_states_to_index (int s1, int s2, int s3, int s4)
- void adp_xor3_sf (gsl_matrix ∗A[2][2][2][2])
- double adp_xor3 (gsl_matrix ∗A[2][2][2][2], uint32_t da, uint32_t db, uint32_t dc, uint32_t dd)
- double adp_xor3_exper (const uint32_t da, const uint32_t db, const uint32_t dc, const uint32_t dd)

7.44.1 Detailed Description

The ADD differential probability of XOR with three inputs ($3\oplus$): $\mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.44.2 Function Documentation

7.44.2.1 double adp_xor3 (gsl_matrix ∗ *A[2][2][2][2],* uint32_t *da,* uint32_t *db,* uint32_t *dc,* uint32_t *dd*)

The additive differential probability (ADP) of $\mathrm{adp}^{3\oplus}$. **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ computed with adp_xor3_sf.
da	first input difference.
db	second input difference.
dc	third input difference.
dd	output difference.

Returns

$\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

See also

adp_xor

7.44.2.2 void adp_xor3_alloc_matrices (gsl_matrix ∗ *A[2][2][2][2]*)

Allocate memory for the transition probability matrices for $\mathrm{adp}^{3\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

See also

adp_xor3_free_matrices

7.44.2.3 double adp_xor3_exper (const uint32_t *da,* const uint32_t *db,* const uint32_t *dc,* const uint32_t *dd*)

The additive differential probability (ADP) of $\mathrm{adp}^{3\oplus}$ computed experimentally over all inputs. **Complexity:** $O(2^{3n})$.

Parameters

da	first input difference.
db	second input difference.

dc	third input difference.
dd	output difference.

Returns

$\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

See also

adp_xor

7.44.2.4 void adp_xor3_free_matrices (gsl_matrix ∗ *A[2][2][2][2]*)

Free memory reserved by a previous call to adp_xor3_alloc_matrices.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.44.2.5 void adp_xor3_normalize_matrices (gsl_matrix ∗ *A[2][2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.44.2.6 void adp_xor3_print_matrices (gsl_matrix ∗ *A[2][2][2][2]*)

Print the matrices for $\mathrm{adp}^{3\oplus}$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.44.2.7 void adp_xor3_print_matrices_sage (gsl_matrix ∗ *A[2][2][2][2]*)

Print the matrices for $\mathrm{adp}^{3\oplus}$ in a format readable by the computer algebra system Sage (http://www.sagemath.org/).

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$.

7.44.2.8 void adp_xor3_sf (gsl_matrix ∗ *A[2][2][2][2]*)

S-function for $\mathrm{adp}^{3\oplus}$: $\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$.

$A[2][2][2][2] = A[da[i]][db[i]][dc[i]][dd[i]]$, where

- $da[i]$: the i-th bit of the first input difference.
- $db[i]$: the i-th bit of the second input difference.
- $dc[i]$: the i-th bit of the third input difference.
- $dd[i]$: the i-th bit of the output difference.

See also

adp_xor_sf

7.44.2.9 int **adp_xor3_states_to_index** (int *s1*, int *s2*, int *s3*, int *s4*)

Transform the values of the four states of the S-function for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf) into an index.

Parameters

s1	state corresponding to the first input difference.
s2	state corresponding to the second input difference.
s3	state corresponding to the third input difference.
s4	state corresponding to the output difference.

Returns

the index $i = (s_4 + 1)2^3 + s_3 2^2 + s_2 2 + s_1$

7.45 src/adp-xtea-f-fk.cc File Reference

The ADD differential probability of the F-function of XTEA for a fixed key and round constants $\mathrm{adp}^F(k, \delta|\, da \rightarrow dd)$. Complexity: $O(n) < c \leq O(2^n)$.

```
#include "common.hh" #include "adp-xor.hh" #include "max-adp-xor.-
hh" #include "adp-xor-fi.hh" #include "max-adp-xor-fi.hh"
#include "adp-shift.hh" #include "xtea.hh"
```

Functions

- double adp_xtea_f_exper (const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double adp_xtea_f_approx (const uint32_t ninputs, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double max_dy_adp_xtea_f_exper (const uint32_t dx, uint32_t ∗dy_max, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_-const)
- double max_dx_adp_xtea_f_exper (uint32_t ∗dx_max, const uint32_t dy, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_-const)
- double adp_xtea_f_lxr_exper (const uint32_t da, const uint32_t db, uint32_t lsh-_const, uint32_t rsh_const)
- double adp_xtea_f_lxr_approx (const uint32_t ninputs, const uint32_t da, const uint32_t db, uint32_t lsh_const, uint32_t rsh_const)
- bool adp_xtea_f_lxr_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool adp_xtea_f_lxr_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, int32_t x)
- uint32_t adp_xtea_f_lxr_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_-const, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double adp_xtea_f_lxr (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t lsh_const, const uint32_t rsh_const)
- double adp_xtea_f_approx (const uint32_t n, gsl_matrix ∗A[2][2][2], const uint32-_t dx, const uint32_t dy, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- bool adp_xtea_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool adp_xtea_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- uint32_t adp_xtea_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- uint32_t adp_xtea_f_assign_bit_x_dx (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_-const, const uint32_t key, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dx)
- uint32_t adp_xtea_f_assign_bit_x_dy (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_-const, const uint32_t key, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dy)

- double adp_xtea_f (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_-const)
- double max_dy_adp_xtea_f (const uint32_t n, const uint32_t dx, uint32_t ∗ret_dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double max_dx_adp_xtea_f (const uint32_t n, uint32_t ∗ret_dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double first_nz_adp_xtea_f (gsl_matrix ∗A[2][2][2], gsl_matrix ∗AA[2][2][2], const uint32_t key, const uint32_t delta, const uint32_t da, uint32_t ∗ret_dd, uint32_t lsh_const, uint32_t rsh_const)

7.45.1 Detailed Description

The ADD differential probability of the F-function of XTEA for a fixed key and round constants $\mathrm{adp}^F(k,\delta|\, da \rightarrow dd)$. Complexity: $O(n) < c \leq O(2^n)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Attention

The algorithms in this file have complexity that depends on the input and output differences to F. It is worst-case exponential in the word size, but is sub-exponential on average.

See also

xdp-xtea-f-fk.cc

7.45.2 Function Documentation

7.45.2.1 double adp_xtea_f (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher XTEA: $\mathrm{adp}^F(k,\delta|\, dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k, \delta|\, dx \rightarrow dy)$.

See also

adp_f_assign_bit_x

7.45.2.2 double adp_xtea_f_approx (const uint32_t *ninputs*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the ADP of the XTEA F-function (xtea_f) obtained over a number of input chosen plaintext pairs chosen uniformly at random.

Parameters

ninputs	number of chosen plaintext pairs.
da	input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k, \delta|\, dx \rightarrow dy)$.

Note

For the exact computation refer to adp_xtea_f

7.45.2.3 double adp_xtea_f_approx (const uint32_t *n*, gsl_matrix ∗ *A[2][2][2]*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the ADD differential probability (ADP) of the XTEA F-function (xtea-_f) with fixed round key and round cnstant, obtained as the multiplication the ADP of its f_{LXR} component (adp_xtea_f_lxr) and the ADP of XOR with one fixed input (adp_xor_-fixed_input):

$\mathrm{adp}^F(k, \delta|\, dx \rightarrow dy) = \mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dt) \cdot \mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dx+dt \rightarrow dy)$.

Algorithm sketch:

1. Compute dz s.t. $p_1 = \max_{dz} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dy \rightarrow dz)$.

Note

Note that $\mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dy \rightarrow dz) = \mathrm{adp}^{\oplus}_{\mathrm{FI}}(k+\delta, dz \rightarrow dy)$.

2. Compute the output from $f_{\mathrm{LXR}} : dt = dz - dx$.
3. Compute $p_2 = \mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dt)$.
4. Compute $\mathrm{adp}^{F}(k, \delta |\ dx \rightarrow dy) = p_1 \cdot p_2$.

Parameters

n	word size.
A	transition probability matrices for $\mathrm{adp}^{\oplus}$ with FI (adp_xor_fixed_input_-sf).
dx	input difference.
dy	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}} \cdot \mathrm{adp}^{\oplus}_{\mathrm{FI}}$.

Note

For the exact computation refer to adp_xtea_f.

7.45.2.4 uint32_t adp_xtea_f_assign_bit_x (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the F-function of XTEA is satisfied. The function operates by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_-xtea_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i + rsh_const`).
key	round key.
delta	round constant.
lsh_const	LSH constant.

rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key ADD probability of `F`: $\mathrm{adp}^F(k, \delta \vert\ dx \rightarrow dy)$.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

adp_f_fk

7.45.2.5 uint32_t adp_xtea_f_assign_bit_x_dx (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t $*$ *x_cnt*, double $*$ *ret_prob*, uint32_t $*$ *ret_dx*)

For given output difference `dy`, compute all input differences `dx` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability input difference.

The function works by recursively assigning the bits of `x` and `dx` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_f_is_sat .

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dx`.
ret_prob	the maximum probability over all input differences $\max_{dx} \mathrm{adp}^F(k, \delta \vert\ dx \rightarrow dy)$.
ret_dx	the input difference that has maximum probability.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_f_assign_bit_x

7.45.2.6 uint32_t **adp_xtea_f_assign_bit_x_dy** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dy*)

For given input difference dx, compute all output differences dy and their probabilities, by counting all values x that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability output difference.

The function works by recursively assigning the bits of x and dy starting at bit position i and terminating at the MS bit n. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the i LS bits. This is checked by applying adp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the i LS bits of x.
x	input value of size at least (i + rsh_const).
lsh_const	LSH constant.
rsh_const	RSH constant.
key	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every dy.
ret_prob	the maximum probability over all output differences $\max_{dy} \mathrm{adp}^F(k, \delta \mid dx \rightarrow dy)$.
ret_dy	the output difference that has maximum probability.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_f_assign_bit_x_dx

7.45.2.7 bool adp_xtea_f_check_x (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value x satisfies the ADD differential $(dx \rightarrow dy)$ for the XTEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k, \delta: \; dy = F(x + dx) - F(x)$.

7.45.2.8 double adp_xtea_f_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant ADD differential probability of the F-function of block cipher XTEA: $\mathrm{adp}^F(k, \delta | \, dx \rightarrow dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^F(k, \delta |\ dx \rightarrow dy)$.

7.45.2.9 bool adp_xtea_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` (xtea_f) is satisfied on the `i` LS bits of `x` i.e. check if

$k, \delta :\ dy[i-1:0] = F(x[i-1:0] + dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i` + `rsh_const`).

Returns

TRUE if $k, \delta :\ dy[i-1:0] = F(x[i-1:0] + dx[i-1:0]) - F(x[i-1:0]) \bmod 2^i$.

7.45.2.10 double adp_xtea_f_lxr (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the ADD differential probability of the f_{LXR} (xtea_f_lxr) function- : $\mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dy)$. **Complexity** c: $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dy)$.

See also

adp_xtea_f_lxr_assign_bit_x

7.45.2.11 double adp_xtea_f_lxr_approx (const uint32_t *ninputs*, const uint32_t *da*, const uint32_t *db*, uint32_t *lsh_const*, uint32_t *rsh_const*)

An approximation of the ADP of f_{LXR} (xtea_f_lxr) obtained over a number of input chosen plaintext pairs chosen uniformly at random.

Parameters

ninputs	number of input chosen plaintext pairs.
da	input difference.
db	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}}(da \rightarrow db)$

Note

For the exact computation refer to adp_xtea_f_lxr_exper

7.45.2.12 uint32_t adp_xtea_f_lxr_assign_bit_x (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the f_{LXR} (xtea-_f_lxr) function is satisfied. The algorithm works by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying adp_xtea_f_lxr_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the probability $\mathrm{adp}^{f_{\mathrm{LXR}}}(dx \rightarrow dy)$.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

adp_xtea_f_lxr

7.45.2.13 bool adp_xtea_f_lxr_check_x (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value x satisfies the ADD differential $(dx \rightarrow dy)$ for the function f_{LXR} (xtea_f_lxr).

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $dy = f_{\mathrm{LXR}}(x+dx) - f_{\mathrm{LXR}}(x)$.

7.45.2.14 double adp_xtea_f_lxr_exper (const uint32_t *da*, const uint32_t *db*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the ADD differential probability of the f_{LXR} (xtea_f_lxr) component of the - F-function of block cipher XTEA, through exhaustive search over all input values. - **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{f_{\mathrm{LXR}}}(da \rightarrow db)$

7.45.2.15 bool adp_xtea_f_lxr_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, int32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for the function f_{LXR} (xtea_f_lxr) is satisfied on the i LS bits of x i.e. check if

$dy[i-1:0] = f_{\mathrm{LXR}}(x[i-1:0] + dx[i-1:0]) - f_{\mathrm{LXR}}(x[i-1:0]) \bmod 2^i$.

Attention

x must be of size at least $(i+R)$ bits where R is the RSH constant of f_{LXR}.

Parameters

mask_i	i bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x	input value of size at least (i + rsh_const).

Returns

TRUE if $dy[i-1:0] = f_{\mathrm{LXR}}(x[i-1:0] + dx[i-1:0]) - f_{\mathrm{LXR}}(x[i-1:0]) \bmod 2^i$.

7.45.2.16 double first_nz_adp_xtea_f (gsl_matrix ∗ *A[2][2][2]*, gsl_matrix ∗ *AA[2][2][2]*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *da*, uint32_t ∗ *ret_dd*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For the XTEA F-function (xtea_f), for fixed input difference da, compute an arbitrary dd such that the differential $(da \rightarrow dd)$ has non-zero probability.

The procedure approximates the ADP of the TEA F-function as a multiplication of the A-DP of its three non-linear components (w.r.t. ADD differences): the two XOR operations and the RSH operation (see xtea_f):

$\mathrm{adp}^F(k,\delta |\ dx \rightarrow dy) = \mathrm{adp}^{\oplus} \cdot \mathrm{adp}^{\gg} \cdot \mathrm{adp}^{\oplus}_{\mathrm{FI}}$

Algorithm sketch:

1. Compute $dy : \max_{dc[i]} \mathrm{adp}^{\oplus}(db, dc[i] \rightarrow dy)$, where $dc[i] \in \{(da \gg 5), (da \gg 5)+1, (da \gg 5) - 2^{n-5}, (da \gg 5) - 2^{n-5}+1\}$, is one of the four possible ADD differences after RSH (adp_rsh).
2. Compute $dt = dy + da$.
3. Compute $dd : \max_{dd} \mathrm{adp}^{\oplus}((k+\delta), dt \rightarrow dd)$.
4. For the computed da and dd experimenttaly re-adjust the probability using adp-_xtea_f_approx.

   Note

   At this step the *exact* probability can also be computed with adp_xtea_f which is more accurate but less efficient.

5. Return the adjusted probability p and dd.

Attention

it is still possible that p = 0.0 for some `da`.

Parameters

A	transition probability matrices for $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for $\mathrm{adp}^{\oplus}$ with FI (adp_xor_fixed_input_-sf).
key	round key.
delta	round constant.
da	input difference.
ret_dd	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{adp}^{F}(k,\delta|\ da \rightarrow dd)$

7.45.2.17 double **max_dx_adp_xtea_f** (const uint32_t *n*, uint32_t ∗ *ret_dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{adp}^{F}(k,\delta|\ dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

n	word size.
ret_dx	maximum probability input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^{F}(k,\delta|\ dx \rightarrow dy)$.

See also

max_dx_adp_f_fk

7.45.2.18 double max_dx_adp_xtea_f_exper (uint32_t ∗ *dx_max*, const uint32_t *dy*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{adp}^F(k, \delta | \; dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx_max	maximum probability input difference.
dy	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^F(k, \delta | \; dx \rightarrow dy)$.

See also

max_dx_adp_f_fk

7.45.2.19 double max_dy_adp_xtea_f (const uint32_t *n*, const uint32_t *dx*, uint32_t ∗ *ret_dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{adp}^F(k, \delta | \; dx \rightarrow dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory requirement:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^{F}(k, \delta |\ dx \rightarrow dy)$.

See also

adp_f_assign_bit_x_dy, max_dx_adp_f_fk

7.45.2.20 double max_dy_adp_xtea_f_exper (const uint32_t *dx*, uint32_t * *dy_max*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{adp}^{F}(k, \delta |\ dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx	input difference.
dy_max	maximum probability output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{adp}^{F}(k, \delta |\ dx \rightarrow dy)$.

See also

max_dy_adp_f_fk

7.46 src/common.cc File Reference

Common functions used accross all YAARX programs.

```
#include "common.hh"
```

Functions

- uint32_t random32 ()
- uint32_t hw8 (uint32_t x)
- uint32_t hw32 (uint32_t x)
- bool is_even (uint32_t i)
- uint32_t gen_sparse (uint32_t hw, uint32_t n)
- void print_binary (uint32_t n)
- bool operator< (differential_t x, differential_t y)

- bool operator== (differential_t a, differential_t b)
- void print_set (const std::set< differential_t, struct_comp_diff_dx_dy > diff_set_-dx_dy)
- void print_mset (const std::multiset< differential_t, struct_comp_diff_p > diff_-mset_p)

7.46.1 Detailed Description

Common functions used accross all YAARX programs.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.46.2 Function Documentation

7.46.2.1 uint32_t gen_sparse (uint32_t *hw*, uint32_t *n*)

Generate a random sparse n-bit difference with Hamming weight hw.

7.46.2.2 uint32_t hw32 (uint32_t *x*)

Hamming weight of a 32-bit word.

7.46.2.3 uint32_t hw8 (uint32_t *x*)

Hamming weight of a byte.

7.46.2.4 bool is_even (uint32_t *i*)

Returns true if the argument is an even number.

7.46.2.5 bool operator< (differential_t *x*, differential_t *y*)

Compare two differentials by probability.

7.46.2.6 bool operator== (differential_t *a*, differential_t *b*)

Evaluate if two differentials are identical. Returns TRUE if they are.

7.46.2.7 void print_binary (uint32_t *n*)

Print a value in binary.

7.46.2.8 void print_mset (const std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p*)

Print the list of 2d differentials stored represented as an STL multiset and ordered by probability.

7.46.2.9 void print_set (const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

Print the list of 2d differentials stored represented as an STL set and ordered by index idx = ((2^n dx) + dy), where n is the word size.

7.46.2.10 uint32_t random32 ( )

Generate a random 32-bit value.

7.47 src/eadp-tea-f-program.cc File Reference

The probability $\mathrm{eadp}^F(da \rightarrow dd)$ with user-provided input.

`#include "common.hh" #include "adp-xor3.hh" #include "tea.hh" #include "eadp-tea-f.hh"`

Functions

- void **eadp_tea_f_program** ()
- int main ()

7.47.1 Detailed Description

The probability $\mathrm{eadp}^F(da \rightarrow dd)$ with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.47.2 Function Documentation

7.47.2.1 int main ( )

Main function for the EADP-TEA-F program.

7.48 src/eadp-tea-f.cc File Reference

The expected additive differential probability (EADP) of the F-function of TEA, averaged over all round keys and constants: $\mathrm{eadp}^F(da \rightarrow dd)$. Complexity: $O(n)$.

```
#include "common.hh" #include "adp-xor3.hh" #include "max-adp-xor3-set.-
hh" #include "adp-shift.hh" #include "tea.hh" #include
"eadp-tea-f.hh"
```

Functions

- double eadp_tea_f_exper (const uint32_t dx, const uint32_t dy, uint32_t lsh_-const, uint32_t rsh_const)
- double eadp_tea_f (gsl_matrix *A[2][2][2][2], const uint32_t da, const uint32_t db, double *prob_db, uint32_t lsh_const, uint32_t rsh_const)
- double max_eadp_tea_f (gsl_matrix *A[2][2][2][2], const uint32_t da, uint32_t *dd_max, double *prob_max, uint32_t lsh_const, uint32_t rsh_const)
- double max_eadp_tea_f_exper (gsl_matrix *A[2][2][2][2], const uint32_t da, uint32_t *dd_max, double *prob_max, uint32_t lsh_const, uint32_t rsh_const)
- void nz_eadp_tea_f_i (const uint32_t k, const uint32_t n, gsl_matrix *A[2][2][2][2], gsl_vector *C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_-t *dd, double *p, double *p_thres, uint32_t *ret_dd, double *ret_p, uint32_t *cnt, uint32_t max_cnt)
- double nz_eadp_tea_f (gsl_matrix *A[2][2][2][2], uint32_t da, uint32_t *ret_dd)

7.48.1 Detailed Description

The expected additive differential probability (EADP) of the F-function of TEA, averaged over all round keys and constants: $\mathrm{eadp}^F(da \rightarrow dd)$. Complexity: $O(n)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.48.2 Function Documentation

7.48.2.1 double eadp_tea_f (gsl_matrix * *A[2][2][2][2],* const uint32_t *da,* const uint32_t *db,* double * *prob_db,* uint32_t *lsh_const,* uint32_t *rsh_const*)

Computing the expected additive differential probability (EADP) of the F-function of TE-A, averaged over all round keys and constants. For fixed input and output differences resp. `da` and `db`, it is defined as:

$$\mathrm{eadp}^F(da \rightarrow db) = 2^{-4n} \{\#(k_0, k_1, \delta, x) : F(x+da) - F(x) = db\}.$$

Complexity: $O(n)$.

Algorithm sketch: eadp^F is computed as the multiplication of ADP-s of the two non-linear (w.r.t. XOR differences) components of F, namely XOR and LSH:

$$\mathrm{eadp}^F(da \rightarrow db) = (\sum_{i=0}^{3}(\mathrm{adp}^{\gg 5}(da, dc_i))) \cdot \mathrm{adp}^{3\oplus}_{\mathrm{SET}}((da \ll 4), da, \{dc_0, dc_1, dc_2, dc_3\} \rightarrow db)$$

where $dc_i \in \{(da \gg 5), (da \gg 5) + 1, (da \gg 5) - 2^{n-5}, (da \gg 5) - 2^{n-5} + 1\}$ are the four possible ADD differences after RSH (see adp_rsh) and $\mathrm{adp}^{3\oplus}_{\mathrm{SET}}$ is the ADP of XOR with three inputs where one of the inputs may satisfy any difference from a given *set* (max_adp_xor3_set).

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	input difference.
db	output difference.
prob_db	the expected DP of F.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{eadp}^F(da \rightarrow db)$.

7.48.2.2 double eadp_tea_f_exper (const uint32_t *dx*, const uint32_t *dy*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Computing the expected additive differential probability (EADP) of the F-function of TEA (see eadp_tea_f), experimentally over all round keys and constants.

Complexity: $O(2^{4n})$.

Parameters

dx	input difference.
dy	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{eadp}^F(da \rightarrow db)$.

See also

eadp_tea_f

7.48.2.3 double max_eadp_tea_f (gsl_matrix ∗ *A[2][2][2][2]*, const uint32_t *da*, uint32_t ∗ *dd_max*, double ∗ *prob_max*, uint32_t *lsh_const*, uint32_t *rsh_const*)

For fixed input difference `da`, compute an output difference `dd` that has maximum expected additive differential probability (EADP) averaged over all round keys and constants of the F-function of TEA:

$\max_{dd} \mathrm{eadp}^F(da \rightarrow dd) = 2^{-4n} \{\#(k_0, k_1, \delta, x) : F(x+da) - F(x) = dd\}$.

Complexity: $O(n)$.

Algorithm sketch: eadp^F is computed as the multiplication of ADP-s of the two non-linear (w.r.t. XOR differences) components of F, namely XOR and LSH:

$$\mathrm{eadp}^F(da \rightarrow dd) = (\sum_{i=0}^{3}(\mathrm{adp}^{\gg 5}(da, dc_i))) \cdot \max_{dd} \mathrm{adp}^{3\oplus}_{\mathrm{SET}}((da \ll 4), da, \{dc_0, dc_1, dc_2, dc_3\} \rightarrow dd)$$

where $dc_i \in \{(da \gg 5), (da \gg 5)+1, (da \gg 5) - 2^{n-5}, (da \gg 5) - 2^{n-5} + 1\}$ are the four possible ADD differences after RSH (see adp_rsh) and $\max_{dd} \mathrm{adp}^{3\oplus}_{\mathrm{SET}}$ is the maximum ADP over all outpt differences, of XOR with three inputs where one of the inputs may satisfy any difference from a given *set* (max_adp_xor3_set).

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	input difference.
dd_max	maximum probability output difference.
prob_max	maximum expected DP of F over all output differences.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{db} \mathrm{eadp}^F(da \rightarrow dd)$.

7.48.2.4 double max_eadp_tea_f_exper (gsl_matrix ∗ *A[2][2][2][2]*, const uint32_t *da*, uint32_t ∗ *dd_max*, double ∗ *prob_max*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Computing the maximum expected additive differential probability (EADP) of the F-function of TEA (see eadp_tea_f), experimentally over all round keys, round constants and output differences.

Complexity: $O(2^{5n})$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	input difference.
dd_max	output difference.
prob_max	the maximum expected DP of F.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{eadp}^F(da \rightarrow db)$.

See also

max_eadp_tea_f

7.48.2.5 double nz_eadp_tea_f (gsl_matrix ∗ *A[2][2][2][2]*, uint32_t *da*, uint32_t ∗ *ret_dd*)

For fixed input diffference `da` to the TEA F-function, generate an arbitrary output difference `dd` for which the expected DP of F is nonzero i.e. $\mathrm{eadp}^F(da \rightarrow dd) > 0$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
da	first input difference to XOR3.
ret_dd	output difference that is returned as result.

Returns

$\mathrm{eadp}^F(da \rightarrow dd)$.

Attention

Although the resulting differential $(da \rightarrow dd)$ is guaranteed to have expected probability, averaged over all keys and constants, strictly bigger than zero, its probability may still be zero for some fixed value of the round keys and δ constants.

See also

nz_eadp_tea_f_i

7.48.2.6 void nz_eadp_tea_f_i (const uint32_t *k*, const uint32_t *n*, gsl_matrix ∗ *A[2][2][2][2]*, gsl_vector ∗ *C*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t ∗ *dd*, double ∗ *p*, double ∗ *p_thres*, uint32_t ∗ *ret_dd*, double ∗ *ret_p*, uint32_t ∗ *cnt*, uint32_t *max_cnt*)

For fixed input diffferences `da`, `db` and `dc`, to the XOR operation with three inputs in the TEA F-function, generate an arbitrary output difference `dd` for which the expected DP of F is nonzero i.e. $\mathrm{eadp}^F(da \rightarrow dd) > 0$.

Complexity c: $O(n) \leq c \ll O(2^n)$.

Algorithm sketch:

The function works recursively starting from the LS bit `k` = 0 and terminating at the MS bit `n`. At every bit position i it assigns values to the i-th bit of the output difference `dd` and evaluates the probability of the resulting partial (i+1)-bit differential: $(da[i:0], db[i:0], dc[i:0] \rightarrow dd[i:0])$. The recursion proceeds only if this probability is not less than the threshold `p_thres`. When i = n, the difference $dd[n-1:0]$ is stored as the result and the probability $\mathrm{eadp}^F(da \rightarrow dd)$ is returned.

Note

Note that the threshold `p_thres` is initialized to 0.0, but is dynamically updated during the execution as soon as a higher value is found.

Attention

Although the resulting differential $(da \rightarrow dd)$ is guaranteed to have expected probability, averaged over all keys and constants, strictly bigger than zero, its probability may still be zero for some fixed value of the round keys and δ constants.

Parameters

k	current bit position in the recursion.
n	word size.
A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
C	unit column vector for computing $\mathrm{adp}^{3\oplus}$ (adp_xor3).
da	first input difference to XOR3.
db	second input difference to XOR3.
dc	third input difference to XOR3.
dd	output difference from XOR3 (and F).
p	probability of the differential $(da[k:0], db[k:0], dc[k:0] \rightarrow dd[k:0])$.
p_thres	probability threshold.
ret_dd	output difference that is returned as result.
ret_p	the EDP $\mathrm{eadp}^F(da \rightarrow dd)$.
cnt	number of output differences generated so far.
max_cnt	maximum number of output differences allowed (typically 1).

See also

adp_xor_ddt

7.49 src/max-adp-xor-fi-program.cc File Reference

The probability ($\max_{dc} \mathrm{adp}^{\oplus}(a, db \rightarrow dc)$) with user-provided input.

`#include "common.hh"` `#include "adp-xor-fi.hh"` `#include "max-adp-xor-fi.hh"`

Functions

- void max_adp_xor_fixed_input_program ()
- int main ()

7.49.1 Detailed Description

The probability ($\max_{dc} \mathrm{adp}^{\oplus}(a, db \rightarrow dc)$) with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.49.2 Function Documentation

7.49.2.1 int main ( )

Main function for the MAX-ADP-XOR-FI program.

7.49.2.2 void max_adp_xor_fixed_input_program ( )

Compute MAX-ADP-XOR-FI with user-provided input.

7.50 src/max-adp-xor-fi.cc File Reference

The maximum ADD differential probability of XOR with one fixed input: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow dc)$.

`#include "common.hh"` `#include "max-adp-xor.hh"` `#include "adp-xor-fi.hh"`

Functions

- double max_adp_xor_fixed_input (gsl_matrix ∗A[2][2][2], const uint32_t a, const uint32_t db, uint32_t ∗dd_max)
- double max_adp_xor_fixed_input_exper (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dc_max)

7.50.1 Detailed Description

The maximum ADD differential probability of XOR with one fixed input: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(a, db \rightarrow dc)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.50.2 Function Documentation

7.50.2.1 double max_adp_xor_fixed_input (gsl_matrix ∗ *A[2][2][2]*, const uint32_t *a*, const uint32_t *db*, uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(da, db \rightarrow dc)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
a	input value.
db	input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(da, db \rightarrow dc)$.

See also

max_adp_xor_bounds, max_adp_xor_i

7.50.2.2 double max_adp_xor_fixed_input_exper (gsl_matrix ∗ *A[2][2][2]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dc_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	input value.
db	input difference.
dc_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}_{\mathrm{FI}}(da, db \rightarrow dc)$.

See also

max_adp_xor_fixed_input

7.51 src/max-adp-xor-program.cc File Reference

Maximum ADD differential probability of XOR ($\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$) with user-provided input.

`#include "common.hh" #include "adp-xor.hh" #include "max-adp-xor.hh"`

Functions

- void max_adp_xor_program ()
- int main ()

7.51.1 Detailed Description

Maximum ADD differential probability of XOR ($\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$) with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.51.2 Function Documentation

7.51.2.1 int main ( )

Main function for the MAX-ADP-XOR program.

7.51.2.2 void max_adp_xor_program ( )

Compute MAX-ADP-XOR with user-provided input.

7.52 src/max-adp-xor.cc File Reference

The maximum ADD differential probability of XOR: $\max_{dc} \mathrm{adp}^{\oplus}(da, db \rightarrow dc)$.

`#include "common.hh" #include "adp-xor.hh"`

Functions

- void max_adp_xor_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C, const uint32_t da, const uint32_t db, uint32_t ∗dd_max, double ∗p_max, uint32_t A_size)
- void max_adp_xor_bounds (gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], const uint32_t da, const uint32_t db, uint32_t ∗dd_max, uint32_t A_size)
- double max_adp_xor (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dd_max)
- double max_adp_xor_exper (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dc_max)

7.52.1 Detailed Description

The maximum ADD differential probability of XOR: $\max_{dc} \mathrm{adp}^{\oplus}(da,db \rightarrow dc)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.52.2 Function Documentation

7.52.2.1 double max_adp_xor (gsl_matrix ∗ *A[2][2][2],* const uint32_t *da,* const uint32_t *db,* uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{adp}^{\oplus}(da,db \rightarrow dc)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}(da,db \rightarrow dc)$.

See also

max_adp_xor_bounds, max_adp_xor_i

7.52.2.2 void max_adp_xor_bounds (gsl_matrix * *A[2][2][2]*, gsl_vector * *B[WORD_SIZE+1]*, const uint32_t *da*, const uint32_t *db*, uint32_t * *dd_max*, uint32_t *A_size*)

Compute an array of bounds that can be used in the computation of the maximum differential probability.

Parameters

A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

Algorithm Outline:

- Initialize $B[n][i] \leftarrow 1, \ \forall i: \ 0 \leq i < (A_{\text{size}} - 1)$
- For every bit position k from n-1 down to 0
  - For every state `i` from 0 to (`A_size` - 1)
    - Initialize $B[k][i] \leftarrow p_{\max} = 0$
    - Let C^i_{k-1} be a column unit vector of size `A_size` with 1 at position i
    - Recursively assign values to the bits of the output difference `dc` starting at bit popsition $j = k$ and terminating at bit position `n`.
      - The recursion proceeds to bit postion $j+1$ only if the probability p_j of the partially constructed differential $(da[j:k], db[j:k] \rightarrow dc[j:k])$ multiplied by the bound of the probability until the end $B[j+1]$ is bigger than the best probability found so far i.e. if: $B[j+1]A_jA_{j-1}\ldots A_kC^i_{k-1} > p_{\max}$
      - When $j = n$ update the max.: $p_{\max} \leftarrow p_{n-1} = \mathrm{dp}(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$.
    - At the end of the recursion set $B[k][i] \leftarrow p_{\max}$.

Meaning of the bounds B:

$B[k][i]$ is an *upper bound* on on the maximum probability of the differential $(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$ because clearly for any choice $dc[n-1:k]$ of the $(n-k)$ MS bits of dc, the probability $LA_{n-1}A_{n-2}\ldots A_kC^i_{k-1}$ will never be bigger than $B[k][i]$. Furthermore, let $G[k] = LA_{n-1}A_{n-2}\ldots A_k$ be the multiplication of the corresponding transition probability matrices for the $(n-k)$ MS bits of dc $dc[n-1:k]$

and let $H[k-1] = A_{k-1}A_{k-2}\ldots A_0C^i_{k-1}$ and $H[-1] = C$. Then $\mathrm{dp}(da, db \to dc) = G[k]H[k-1] \leq B[k]H[k-1]$ for *any* choice of $dc[n-1:k]$. In particular, when $k = 0$ $\mathrm{dp}(da, db \to dc) = \max_{dc} \mathrm{dp}(da, db \to dc) = G[0]C = B[0]C$.

See also

max_adp_xor_i

7.52.2.3 double max_adp_xor_exper (gsl_matrix ∗ *A[2][2][2]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dc_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{adp}^{\oplus}(da, db \to dc)$.

See also

max_adp_xor

7.52.2.4 void max_adp_xor_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double ∗ *p*, uint32_t ∗ *dd*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, gsl_vector ∗ *C*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*, double ∗ *p_max*, uint32_t *A_size*)

Compute an *upper bound* $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k] \to dc[n-1:k])$ starting from initial state `i` of the S-function i.e. $\mathrm{dp}(da[n-1:k], db[n-1:k] \to dc[n-1:k]) = LA_{n-1}A_{n-2}\ldots A_kC^i_{k-1}$, given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j] \to dc[n-1:j])$ for $j = k+1, k+2, \ldots, n-1$, where $L = [1\ 1\ \ldots\ 1]$ is a row vector of size `A_size` and C^i_{k-1} is a unit column vector of size `A_size` with 1 at position i and $C^i_{-1} = C$.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the estimated probability at bit position `k`.
dd	output difference.

A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

Algorithm Outline:

Recursively assign values to the bits of the output difference `dc` starting at bit popsition $j = k$ and terminating at bit position `n`. The recursion proceeds to bit postion $j+1$ only if the probability p_j of the partially constructed differential $(da[j:k], db[j:k] \rightarrow dc[j:k])$ multiplied by the bound of the probability until the end $B[j+1]$ is bigger than the best probability found so far i.e. if: $B[j+1]A_j A_{j-1} \dots A_k C^i_{k-1} > p_{\max}$. When $j = n$ update the max.: $p_{\max} \leftarrow p_{n-1} = \mathrm{dp}(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$.

See also

max_adp_xor_bounds

7.53 src/max-adp-xor3-program.cc File Reference

The probability $\max_{dd} \mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$ with user-provided input.

`#include "common.hh" #include "adp-xor3.hh" #include "max-adp-xor3.hh"`

Functions

- void **max_adp_xor3_program** ()
- int main ()

7.53.1 Detailed Description

The probability $\max_{dd} \mathrm{adp}^{3\oplus}(da, db, dc \rightarrow dd)$ with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.53.2 Function Documentation

7.53.2.1 int main ( )

Main function for the MAX-ADP-XOR3 program.

7.54 src/max-adp-xor3-set.cc File Reference

The maximum ADD differential probability of XOR with three inputs, where one of the inputs satisfies a *set* of ADD differences: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$.

`#include "common.hh" #include "adp-xor3.hh" #include "max-adp-xor3.hh" #include "max-adp-xor3-set.hh"`

Functions

- void max_adp_xor3_set_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C[ADP_XOR3_SET_SIZE], const uint32_t da, const uint32_t db, const uint32_t dc[ADP_XOR3_SET_SIZE], uint32_t ∗dd_max, double ∗p_max)
- double max_adp_xor3_set (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc[ADP_XOR3_SET_SIZE], double p_dc[ADP_XOR3_SET_SIZE], uint32_t ∗dd_max)
- double max_adp_xor3_set_exper (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc[ADP_XOR3_SET_SIZE], double p_dc[ADP_XOR3_SET_SIZE], uint32_t ∗dd_max)

7.54.1 Detailed Description

The maximum ADD differential probability of XOR with three inputs, where one of the inputs satisfies a *set* of ADD differences: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.54.2 Function Documentation

7.54.2.1 double max_adp_xor3_set (gsl_matrix ∗ *A[2][2][2][2],* const uint32_t *da,* const uint32_t *db,* const uint32_t *dc[ADP_XOR3_SET_SIZE],* double *p_dc[ADP_XOR3_SET_SIZE],* uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences for a set of input differenecs: $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$.

Complexity c: $O(nR) \leq c \leq O(2^{nR})$, where R is the size of the set of input differences dc_r.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	set of input difference.
dd_max	maximum probability output difference.
p_dc	probabilities of the set of differentials corresponding to the set of differences (used for testing and debug only).

Returns

$\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \dots\} \rightarrow dd)$.

Algorithm Outline:

- Compute the bounds for each of the differences in the set *independently* using max_adp_xor_bounds i.e. compute $B_r[k]$ - the bounds ror the R differentials: $\mathrm{dp}(da[n-1:k], db[n-1:k], dc_r[n-1:k] \rightarrow dd[n-1:k])$ corresponding to the r-th input differences dc_r in the set.
- Compute a single array of bounds $B_{\max}$ as the maximum of the bounds $B_r[k]$ at every bit position $0 \leq k \leq n$ for every S-function state $0 \leq i < A_{\mathrm{size}}$: $B_{\max}[k][i] = \max_r B[k][i],\ 0 \leq k \leq n,\ 0 \leq i < A_{\mathrm{size}}$.
- Call max_adp_xor3_set_i with the array of bounds $B_{\max}[k][i]$ to compute the final maximum probability $\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}$.

See also

max_adp_xor3_set_i, max_adp_xor_bounds, max_adp_xor

7.54.2.2 double max_adp_xor3_set_exper (gsl_matrix ∗ *A[2][2][2][2]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc[ADP_XOR3_SET_SIZE]*, double *p_dc[ADP_XOR3_SET_SIZE]*, uint32_t ∗ *dd_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	set of input difference.
dd_max	maximum probability output difference.
p_dc	probabilities of the set of differentials corresponding to the set of differences; normally set to 1 (used for testing and debug only).

Returns

$\max_{dd} \mathrm{adp}^{\oplus}_{\mathrm{SET}}(da, db, \{dc_0, dc_1, \ldots\} \rightarrow dd)$.

See also

max_adp_xor3_set

7.54.2.3 void max_adp_xor3_set_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double ∗ *p*, uint32_t ∗ *dd*, gsl_matrix ∗ *A[2][2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, gsl_vector ∗ *C[ADP_XOR3_SET_SIZE]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc[ADP_XOR3_SET_SIZE]*, uint32_t ∗ *dd_max*, double ∗ *p_max*)

Compute an upper bound $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k], \{dc_0[n-1:k], dc_1[n-1:k], \ldots\} \rightarrow dd[n-1:k])$, starting from initial state `i` of the S-function and given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j], \{dc_0[n-1:j], dc_1[n-1:j], \ldots\} \rightarrow dd[n-1:j])$ for $j = k+1, k+2, \ldots, n-1$, where $\{dc_0[n-1:k], dc_1[n-1:k], \ldots\}$ is a finite set of input differences.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the estimated probability at bit position `k`.
dd	output difference.
A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dc	set of input differences.
dd_max	maximum probability output difference.
p_max	the maximum probability.

Algorithm Outline:

The bound for the set of differences is computed as the sum of the bounds of the differentials obtained from each of the elements of the set: $B[k][i] = \sum_r B_r[k][i]$, where $B_r[k][i]$ is an upper bound on the maximum probability of the differential corresponding to the r-th input difference dc_r i.e. $\mathrm{dp}(da[n-1:k], db[n-1:k], dc_r[n-1:k] \rightarrow dd[n-1:k])$ computed as in max_adp_xor_i.

See also

max_adp_xor3_set, max_adp_xor_i

7.55 src/max-adp-xor3.cc File Reference

The maximum ADD differential probability of XOR with three inputs: $\max_{dd} \mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$.

`#include "common.hh"` `#include "adp-xor3.hh"`

Functions

- void max_adp_xor3_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max, double ∗p_max)
- void max_adp_xor3_bounds (gsl_matrix ∗A[2][2][2][2], gsl_vector ∗B[WORD_SIZE+1], const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)
- double max_adp_xor3 (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)
- void max_adp_xor3_rec_i (const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2][2], gsl_vector ∗C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max, double ∗p_max)
- double max_adp_xor3_rec (gsl_matrix ∗A[2][2][2][2], gsl_vector ∗C, const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)
- double max_adp_xor3_exper (gsl_matrix ∗A[2][2][2][2], const uint32_t da, const uint32_t db, const uint32_t dc, uint32_t ∗dd_max)

7.55.1 Detailed Description

The maximum ADD differential probability of XOR with three inputs: $\max_{dd} \mathrm{adp}^{3\oplus}(da,db,dc \rightarrow dd)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.55.2 Function Documentation

7.55.2.1 double max_adp_xor3 (gsl_matrix ∗ *A[2][2][2][2],* const uint32_t *da,* const uint32_t *db,* const uint32_t *dc,* uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{adp}^{\oplus}(da,db,dc \rightarrow dd)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

See also

max_adp_xor3_bounds, max_adp_xor3_i

7.55.2.2 void max_adp_xor3_bounds (gsl_matrix * *A[2][2][2][2]*, gsl_vector * *B[WORD_SIZE+1]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t * *dd_max*)

Compute an array of bounds that can be used in the computation of the maximum differential probability.

Parameters

A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

See also

max_adp_xor_bounds, max_adp_xor3_i

7.55.2.3 double max_adp_xor3_exper (gsl_matrix * *A[2][2][2][2]*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t * *dd_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dd} \mathrm{adp}^{\oplus}(da, db, dc \rightarrow dd)$

See also

max_adp_xor

7.55.2.4 void max_adp_xor3_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double $*$ *p*, uint32_t $*$ *dd*, gsl_matrix $*$ *A[2][2][2][2]*, gsl_vector $*$ *B[WORD_SIZE+1]*, gsl_vector $*$ *C*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t $*$ *dd_max*, double $*$ *p_max*)

Compute an upper bound $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k], dc[n-1:k] \rightarrow dd[n-1:k])$ starting from initial state `i` of the S-function given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j], dc[n-1:j] \rightarrow dd[n-1:j])$ for $j = k+1, k+2, \ldots, n-1$.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the transition probability of state `i` at bit position `k`.
dd	output difference.
A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.

See also

max_adp_xor_i

7.55.2.5 double max_adp_xor3_rec (gsl_matrix $*$ *A[2][2][2][2]*, gsl_vector $*$ *C*, const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*, uint32_t $*$ *dd_max*)

Recursively compute the maximum differential probability over all output differences: $\max_{dd} \mathrm{adp}^{\oplus}(da, db, dc \rightarrow dd)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
C	unit row vector initialized with 1 at the nitial state.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dd} \mathrm{adp}^{\oplus}(da, db, dc \rightarrow dd)$

Note

This function max_adp_xor3_rec is more efficient than exhaustive search over all output differences max_adp_xor3_exper, but is less efficient than the function max_adp_xor3 that uses bounds. The reason is that at every bit position, max_adp_xor3_rec (by max_adp_xor3_rec_i) implicitly assumes that the remaining probability until the end (i.e. until the MSB) is 1, while the bounds computed by max_adp_xor3 are tighter and thus more branches of the recursion are cut earlier in the computation.

See also: max_adp_xor3_i()

7.55.2.6 void **max_adp_xor3_rec_i** (const uint32_t ***k***, const uint32_t ***n***, double $*$ ***p***, uint32_t $*$ ***dd***, gsl_matrix $*$ ***A[2][2][2][2]***, gsl_vector $*$ ***C***, const uint32_t ***da***, const uint32_t ***db***, const uint32_t ***dc***, uint32_t $*$ ***dd_max***, double $*$ ***p_max***)

Recursively compute the maximum differential probability over all output differences of the partial $(n-k)$-bit differential $\max_{dd} \mathrm{adp}^{\oplus}(da[n-1:k], db[n-1:k], dc[n-1:k] \rightarrow dd[n-1:k])$.

Parameters

k	current bit position: $n > k \geq 0$.
n	word size.
p	the probability at bit position k.
dd	output difference.
A	transition probability matrices.
C	unit row vector initialized with 1 at the nitial state.
da	first input difference.
db	second input difference.
dc	third input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.

Algorithm Outline:

The function recursively assigns the bits of the output difference starting at the LS bit position $k = 0$ and proceeding to $k+1$ only if the probability so far is still above the maximum that was found up to now. The initial value for the maximum probability p_max is 0 and is updated dynamically during the process every time a higher probability is encountered. The recursion stops at the MSB $k = n$.

See also: max_adp_xor3_rec()

7.56 src/max-eadp-tea-f-program.cc File Reference

The probability $\max_{dd} \mathrm{eadp}^F(da \rightarrow dd)$ with user-provided input.

`#include "common.hh"` `#include "adp-xor3.hh"` `#include "tea.hh"` `#include "eadp-tea-f.hh"`

Functions

- void **max_eadp_tea_f_program** ()
- int main ()

7.56.1 Detailed Description

The probability $\max_{dd} \mathrm{eadp}^F(da \rightarrow dd)$ with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.56.2 Function Documentation

7.56.2.1 int main ( )

Main function for the MAX-EADP-TEA-F program.

7.57 src/max-xdp-add-program.cc File Reference

The probability $\max_{dc} \mathrm{xdp}^+(da, db \rightarrow dc)$ with user-provided input.

`#include "common.hh"` `#include "xdp-add.hh"` `#include "max-xdp-add.hh"`

Functions

- void max_xdp_add_program ()
- int main ()

7.57.1 Detailed Description

The probability $\max_{dc} \mathrm{xdp}^+(da, db \rightarrow dc)$ with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.57.2 Function Documentation

7.57.2.1 int main ( )

Main function for the MAX-XDP-ADD program.

7.57.2.2 void max_xdp_add_program ( )

Compute MAX-XDP-ADD with user-provided input.

7.58 src/max-xdp-add.cc File Reference

The maximum XOR differential probability of ADD: $\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$.

`#include "common.hh" #include "xdp-add.hh"`

Functions

- void max_xdp_add_i (const int i, const uint32_t k, const uint32_t n, double ∗p, uint32_t ∗dd, gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], gsl_vector ∗C, const uint32_t da, const uint32_t db, uint32_t ∗dd_max, double ∗p_max, uint32_t A_size)
- void max_xdp_add_bounds (gsl_matrix ∗A[2][2][2], gsl_vector ∗B[WORD_SIZE+1], const uint32_t da, const uint32_t db, uint32_t ∗dd_max, uint32_t A_size)
- double max_xdp_add (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dd_max)
- double max_xdp_add_exper (gsl_matrix ∗A[2][2][2], const uint32_t da, const uint32_t db, uint32_t ∗dc_max)

7.58.1 Detailed Description

The maximum XOR differential probability of ADD: $\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.58.2 Function Documentation

7.58.2.1 double max_xdp_add (gsl_matrix ∗ *A[2][2][2]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*)

Compute the maximum differential probability over all output differences: $\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$. **Complexity** c: $O(n) \leq c \leq O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$.

See also

max_xdp_add, max_xdp_add_i, max_adp_xor

7.58.2.2 void max_xdp_add_bounds (gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *B[WORD_SIZE+1]*, const uint32_t *da*, const uint32_t *db*, uint32_t ∗ *dd_max*, uint32_t *A_size*)

Compute an array of bounds that can be used in the computation of the maximum differential probability.

Parameters

A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

See also

max_xdp_add_i, max_adp_xor_bounds

7.58.2.3 double max_xdp_add_exper (gsl_matrix $*$ *A[2][2][2]*, const uint32_t *da*, const uint32_t *db*, uint32_t $*$ *dc_max*)

Compute the maximum differential probability by exhaustive search over all output differences. **Complexity:** $O(2^n)$.

Parameters

A	transition probability matrices.
da	first input difference.
db	second input difference.
dc_max	maximum probability output difference.

Returns

$\max_{dc} \mathrm{xdp}^{+}(da, db \rightarrow dc)$.

See also

max_xdp_add

7.58.2.4 void max_xdp_add_i (const int *i*, const uint32_t *k*, const uint32_t *n*, double $*$ *p*, uint32_t $*$ *dd*, gsl_matrix $*$ *A[2][2][2]*, gsl_vector $*$ *B[WORD_SIZE+1]*, gsl_vector $*$ *C*, const uint32_t *da*, const uint32_t *db*, uint32_t $*$ *dd_max*, double $*$ *p_max*, uint32_t *A_size*)

Compute an *upper bound* $B[k][i]$ on the maximum probability of the differential $(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k])$ starting from initial state `i` of the S-function i.e. $\mathrm{dp}(da[n-1:k], db[n-1:k] \rightarrow dc[n-1:k]) = LA_{n-1}A_{n-2}\ldots A_k C^i_{k-1}$, given the upper bounds $B[k][i]$ on the probabilities of the differentials $(da[n-1:j], db[n-1:j] \rightarrow dc[n-1:j])$ for $j = k+1, k+2, \ldots, n-1$, where $L = [1\ 1\ \ldots\ 1]$ is a row vector of size `A_size` and C^i_{k-1} is a unit column vector of size `A_size` with 1 at position *i* and $C^i_{-1} = C$.

Parameters

i	index of the state of the S-function: `A_size` $> i \geq 0$.
k	current bit position: $n > k \geq 0$.
n	word size.
p	the estimated probability at bit position `k`.
dd	output difference.
A	transition probability matrices.
B	array of size `A_size` rows by (`n` + 1) columns containing upper bounds on the maximum probabilities of all `j` bit differentials $n \geq j \geq 1$ beginning from any state `i`: `A_size` $> i \geq 0$.

C	unit row vector of size `A_size` rows, initialized with 1 at state index `i`.
da	first input difference.
db	second input difference.
dd_max	maximum probability output difference.
p_max	the maximum probability.
A_size	size of the square transition probability matrices (equivalently, the number of states of the S-function).

See also

max_adp_xor_i

7.59 src/tea-add-ddt-search.cc File Reference

Automatic search for ADD differential trails in TEA using full DDT-s.

`#include "common.hh"` `#include "tea.hh"` `#include "adp-tea-f-fk-ddt.hh"`

Functions

- double verify_trail (uint64_t npairs, differential_t trail[NROUNDS], uint32_t nrounds, uint32_t key[4], uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- void round_ddt (const int n, const int nrounds, differential_t **RSDDT_E, differential_t **RSDDT_O, differential_t *SDDT_O, double B[NROUNDS], double *Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS])
- void tea_search_ddt (uint32_t key[4])
- void round_xddt (const int n, const int nrounds, differential_t ***XRSDDT_E, differential_t ***XRSDDT_O, differential_t **XSDDT_O, const double B[NROUNDS], double *Bn, differential_t diff_in[NROUNDS], differential_t trail[NROUNDS])
- void tea_search_xddt (uint32_t key[4])
- void round_xddt_bottom_up (const int n, const int nrounds, differential_t ***XRSDDT_E, differential_t ***XRSDDT_O, differential_t **XSDDT_E, differential_t **XSDDT_O, const double B[NROUNDS], double *Bn, differential_t diff_in[NROUNDS], differential_t trail[NROUNDS])
- void tea_search_xddt_bottom_up (uint32_t key[4])

7.59.1 Detailed Description

Automatic search for ADD differential trails in TEA using full DDT-s.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Attention

Exponential complexity in the word size; infeasible for word sizes bigger than 11 bits. Used only for tests and verification.

7.59.2 Function Documentation

7.59.2.1 void round_ddt (const int *n*, const int *nrounds*, differential_t ** *RSDDT_E*, differential_t ** *RSDDT_O*, differential_t * *SDDT_O*, double *B[NROUNDS]*, double * *Bn*, const differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*)

Automatic search for ADD differential trails using precomputed full difference distribution tables (DDT) for **a modified version of TEA** that uses the same round constant δ in every round.

Attention

1. Assumes the same δ constant is used at every round of TEA.
2. Two DDT-s are computed: `DDT_E` contains fixed-key probabilities for the round keys applied in all even rounds: $0,2,4,\ldots$; `DDT_O` contains fixed-key probabilities for the round keys applied in all odd rounds: $1,3,5,\ldots$.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
RSDDT_E	a DDT for the keys of all even rounds $0,2,4,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (a Row-Sorted `DDT_E`).
RSDDT_O	a DDT for the keys of all odd rounds $1,3,5,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (a Row-Sorted `DDT_O`).
SDDT_E	a DDT for the keys of all even rounds will all elements sorted in descending order of their probability (a Sorted `DDT_E`).
nrounds	total number of rounds (NROUNDS).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best differential trail for `nrounds`.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$

- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

See also

tea_add_threshold_search

7.59.2.2 void round_xddt (const int *n*, const int *nrounds*, differential_t *** *XRSDDT_E*, differential_t *** *XRSDDT_O*, differential_t ** *XSDDT_O*, const double *B[NROUNDS]*, double * *Bn*, differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*)

Automatic search for ADD differential trails using precomputed full difference distribution tables (DDT) for **the original version of TEA**.

Attention

For every round constant δ, two DDT-s are computed: `DDT_E` containing the fixed-key fixed- δ probabilities for the round keys applied in all even rounds: $0,2,4,\ldots$ and `DDT_O` containing the fixed-key fixed- δ probabilities for the round keys applied in all odd rounds: $1,3,5,\ldots$. Since δ is updated every second round, for N rounds $2(N/2)$ DDT-s will be computed.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
XRSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds $0,2,4,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_E`).
XRSDDT_O	an array of fixed-key fixed- δ DDT-s for all odd rounds $1,3,5,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_O`).
XSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds will all elements sorted in descending order of their probability (an eXtended Sorted `DDT_E`).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best differential trail for `nrounds`.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.

- ...
- $B[i]$: best probability for $(i+1)$ rounds.
- ...
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

See also

round_ddt

7.59.2.3 void round_xddt_bottom_up (const int *n*, const int *nrounds*, differential_t *** *XRSDDT_E*, differential_t *** *XRSDDT_O*, differential_t ** *XSDDT_E*, differential_t ** *XSDDT_O*, const double *B[NROUNDS]*, double * *Bn*, differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*)

Automatic search for ADD differential trails using precomputed full difference distribution tables (DDT) for **the original version of TEA**.

round_xddt_bottom_up is conceptually the same as round_xddt, except that **the search proceeds from the bottom up** i.e. first finds the best 1-round trail for the last round N, next finds the best 2-round trail for rounds $N-1, N$, etc. finds the best i-round trail for rounds $i, i+1, \ldots, N$ and finally finds the best N-round trail.

Attention

For every round constant δ, two DDT-s are computed: `DDT_E` containing the fixed-key fixed- δ probabilities for the round keys applied in all even rounds: $0,2,4,\ldots$ and `DDT_O` containing the fixed-key fixed- δ probabilities for the round keys applied in all odd rounds: $1,3,5,\ldots$. Since δ is updated every second round, for N rounds $2(N/2)$ DDT-s will be computed.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
XRSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds $0,2,4,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_E`).
XRSDDT_O	an array of fixed-key fixed- δ DDT-s for all odd rounds $1,3,5,\ldots$ with the elements in each row (i.e. for a fixed input difference) sorted in descending order of their probability (an eXtended Row-Sorted `DDT_O`).
XSDDT_E	an array of fixed-key fixed- δ DDT-s for all even rounds will all elements sorted in descending order of their probability (an eXtended Sorted `DDT_E`).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.

Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best differential trail for `nrounds`.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for n rounds.
- $B[1]$: best probability for $(n-1)$ rounds.
- ...
- $B[i]$: best probability for $(n-i)$ rounds (rounds $n-i, n-i+1, \ldots, n$).
- ...
- $B[n-2]$: best probability for 2 rounds (rounds $n-1, n$).
- $B[n-1]$: best probability for 1 round (round n).

See also

round_xddt

7.59.2.4 void tea_search_ddt (uint32_t *key[4]*)

Search for ADD differential trails in a modified version of block cipher TEA that uses the same round constant δ in every round. Computes full difference distribution tables (DDT) for every key and the same round constant: a wrapper function for round_ddt.

Parameters

key	cryptographic key of TEA.

Attention

Assumes the same δ constant is used at every round of TEA.

See also

tea_add_trail_search

7.59.2.5 void tea_search_xddt (uint32_t *key[4]*)

Search for ADD differential trails in the original version of block cipher TEA. Computes full difference distribution tables (DDT) for every key and every round constant: a wrapper function for round_xddt.

Parameters

key	cryptographic key of TEA.

See also

round_xddt

7.59.2.6 void tea_search_xddt_bottom_up (uint32_t *key[4]*)

Search for ADD differential trails in the original version of block cipher TEA. Computes full difference distribution tables (DDT) for every key and every round constant. - Conceptually the same as tea_search_xddt, except that the search starts from the last round and proceeds up to the first (i.e. in a bottom-up amnner). This function is a wrapper for round_xddt_bottom_up.

Parameters

key	cryptographic key of TEA.

See also

round_xddt_bottom_up

7.59.2.7 double verify_trail (uint64_t *npairs*, differential_t *trail[NROUNDS]*, uint32_t *nrounds*, uint32_t *key[4]*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Experimentally verify the probabilities of the 1-round differentials composing an N-round differential trail for block cipher TEA, against the exact probabilities from a DDT.

Parameters

npairs	number of chosen plaintext pairs (NPAIRS).
trail	best differential trail for `nrounds`.
nrounds	total number of rounds (NROUNDS).
key	cryptographic key of TEA.
delta	round constant.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).

7.60 src/tea-add-threshold-search.cc File Reference

Automatic search for ADD differential trails in block cipher TEA.

```
#include "common.hh" #include "adp-xor3.hh" #include "tea.-
hh" #include "eadp-tea-f.hh" #include "tea-f-add-pddt.hh"
```

Functions

- void tea_add_threshold_search (const int n, const int nrounds, const uint32_t npairs, const uint32_t key[4], gsl_matrix *A[2][2][2][2], double B[NROUNDS], double *Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS], uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void tea_add_trail_search (uint32_t key[4])

7.60.1 Detailed Description

Automatic search for ADD differential trails in block cipher TEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.60.2 Function Documentation

7.60.2.1 void tea_add_threshold_search (const int *n*, const int *nrounds*, const uint32_t *npairs*, const uint32_t *key[4]*, gsl_matrix * *A[2][2][2][2]*, double *B[NROUNDS]*, double * *Bn*, const differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset< differential_t, struct_comp_diff_p > * *diff_mset_p*, std::set< differential_t, struct_comp_diff_dx_dy > * *diff_set_dx_dy*)

Automatic search for ADD differential trails in block cipher TEA. using pDDT.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
key	cryptographic key of TEA.
A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best found differential trail for `nrounds`.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	set of differentials (dx, dy, p) (the pDDT) ordered by probability p.
diff_set_dx_dy	set of differentials (dx, dy, p) (the pDDT) ordered by index $i = (dx\ 2^n + dy)$.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.

- $B[1]$: best probability for 2 rounds.
- ...
- $B[i]$: best probability for $(i+1)$ rounds.
- ...
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

Algorithm Outline:

The algorithm is based on Matsui search strategy described in [Sect. 4, Matsui, `On correlation between the order of S-boxes and the strength of DES`, EUROCRYPT'94]. The main idea is to view the F-function of TEA as an S-box for which a partial difference distribution table (pDDT) is constructed (tea_f_add_pddt). Then a recursive search for differential trails over a given number of rounds $n \geq 1$ is performed. From knowledge of the best probabilities $B_1, B_2, \ldots, B_{n-1}$ for the first $(n-1)$ rounds and an initial estimate $\overline{B}_n$ for the probability for n rounds the best probability B_n for n rounds is derived. Note that for the estimate the following must hold: $\overline{B}_n \leq B_n$.

In addition to Matsui's notation for the probability of the best n-round trail B_n and of its estimation $\overline{B}_n$ we introduce $\widehat{B}_n$ to denote the probability of *the best found* trail for n rounds: $\overline{B}_n \leq \widehat{B}_n \leq B_n$. Given a pDDT D of maximum size m, an estimation for the best n-round probability $\overline{B}_n$ with its corresponding n-round differential trail $\overline{T}$ and the probabilities $\widehat{B}_1, \widehat{B}_2, \ldots, \widehat{B}_{n-1}$ of the best found trails for the first $(n-1)$ rounds, tea_add_threshold_search outputs an n-round trail $\widehat{T}$ that has probability $\widehat{B_n} \geq \overline{B}_n$.

tea_add_threshold_search operates by recursively extending a trail for i rounds to $(i+1)$ rounds, beginning with $i = 1$ and terminating at $i = n$. This is done by exploring multiple differential trails constructed from the entries of the pDDT D at every round. If, in the process, a differential that is not already in D is encountered it is added to D, provided that the maximum size m has not been reached. The recursion at level i continues to level $(i+1)$ only if the probability of the constructed i-round trail multiplied by the probability of the best found trail for $(n-i)$ rounds is at least $\overline{B}_n$ i.e. if, $p_1 p_2 \ldots p_i \widehat{B}_{n-i} \geq \overline{B}_n$ holds. For $i = n$ the last equation is equivalent to: $p_1 p_2 \ldots p_n = \widehat{B}_n \geq \overline{B}_n$. If the latter holds, the initial estimate is updated with the new: $\overline{B}_n \leftarrow \widehat{B}_n$ and the corresponding trail is also updated accordingly: $\overline{T}_n \leftarrow \widehat{T}_n$. Upon termination the best found trail $\widehat{T}_n$ and its probability $\widehat{B_n}$ are returned as result.

Termination

The algorithm terminates when one of the following two events happens first:

1. The initial estimate $\overline{B}_n$ can not be improved further.
2. The maximum size m of the pDDT D is reached and all differentials in D in every round have been explored.

Complexity

The complexity of tea_add_threshold_search depends on the following factors:

1. The closeness of the best found probabilities $\widehat{B}_1, \widehat{B}_2, \ldots, \widehat{B}_{n-1}$ for the first $(n-1)$ rounds to the actual best probabilities.
2. The tightness of the initial estimate $\overline{B}_n$.
3. The number of elements in D. The latter is determined by the probability threshold used to compute D and by the maximum number of elements m allowed.

In the worst-case, in every round, except the last, m iterations will be executed. - Therefore the worst-case complexity is $\mathcal{O}(m^{n-1})$, where n is the number of rounds. Although the algorithm is worst-case exponential in the number of rounds, it is much more efficient in practice.

Attention

The algorithm does not guarantee to find the *best* trail.

Note

The pDDT of TEA contains the expected differential probabilities of F averaged over all keys and round constants. To obtain better estimate of the probabilities of trails for a fixed key and round constants, in the process of the search the probability of each differential is additionally adjusted to the value of the round key and constant by performing one-round encryptions over `npairs` pairs of chosen plaintexts.

7.60.2.2 void tea_add_trail_search (uint32_t *key[4]*)

Search for ADD differential trails in block cipher TEA: wrapper function for tea_add_-threshold_search.

Parameters

key	cryptographic key of TEA.

Algorithm Outline:

The procedure operates as follows:

1. Compute a pDDT for F (tea_f_add_pddt).
2. Adjust the probabilities of the pDDT to the round key and constant (tea_f_add_-pddt_adjust_to_key).
3. Execute the search for differential trails for n rounds (n = NROUNDS) through a successive application of tea_add_threshold_search :
    - Compute the best found probability on 1 round: $B[0]$.
    - Using $B[0]$ compute the best found probability on 2 rounds: $B[1]$.
    - $\ldots$
    - Using $B[0], \ldots, B[i-1]$ compute the best found probability on $(i+1)$ rounds: $B[i]$.

- ...
- Using $B[0],\ldots,B[n-2]$ compute the best found probability on n rounds: $B[n-1]$.

4. Print the best found trail on n rounds on standrad output and terminate.

See also

tea_add_threshold_search

7.61 src/tea-f-add-pddt.cc File Reference

Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA.

`#include "common.hh" #include "adp-xor3.hh" #include "adp-shift.hh" #include "tea.hh" #include "eadp-tea-f.hh" #include "adp-tea-f-fk.hh"`

Functions

- bool rsh_condition_is_sat (const uint32_t k, const uint32_t new_da, const uint32_t new_dc)
- bool lsh_condition_is_sat (const uint32_t k, const uint32_t new_da, const uint32_t new_db)
- void tea_f_add_pddt_i (const uint32_t k, const uint32_t n, const uint32_t lsh_const, const uint32_t rsh_const, gsl_matrix *A[2][2][2][2], gsl_vector *C, uint32_t *da, uint32_t *db, uint32_t *dc, uint32_t *dd, double *p, const double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void tea_f_add_pddt (uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void tea_f_add_pddt_adjust_to_key (uint32_t nrounds, uint32_t npairs, uint32_t key[4], double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void tea_f_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p, const std::set< differential_t, struct_comp_diff_dx_dy > diff_set_dx_dy)
- void tea_f_add_pddt_exper (gsl_matrix *A[2][2][2][2], uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p)
- void tea_f_add_pddt_fk_exper (uint32_t n, double p_thres, uint32_t delta, uint32_t k0, uint32_t k1, uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p)

7.61.1 Detailed Description

Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.61.2 Function Documentation

7.61.2.1 bool lsh_condition_is_sat (const uint32_t *k*, const uint32_t *new_da*, const uint32_t *new_db*)

Check if two differences da and dc, partially constructed up to bit k (WORD_SIZE $> k \geq 0$), are valid input and output difference respectively, for the LSH operation.

Parameters

k	bit position: WORD_SIZE $> k \geq 0$.
new_da	input difference to LSH partially constructed up to bit k.
new_db	output difference from LSH partially constructed up to bit k.

Returns

TRUE if dc, after being fully constructed, will be a valid output difference from LSH, given the input difference da; FALSE otherwise.

More Details:

1. If $k < L$: check if $db[k:0] = 0$.
2. If $k \geq L$: check if $(db \gg L)[n-(k-L+1):0] = da[n-(k-L+1):0]$.

where L =TEA_LSH_CONST, n =WORD_SIZE.

See also

rsh_condition_is_sat

7.61.2.2 bool rsh_condition_is_sat (const uint32_t *k*, const uint32_t *new_da*, const uint32_t *new_dc*)

Check if two differences da and dc, partially constructed up to bit k (WORD_SIZE $> k \geq 0$), are valid input and output difference respectively, for the RSH operation. From the partial information for dc, the algorithm estimates if dc belongs to one of the four possible differences after the RSH operation (see adp_rsh): $\{(da \gg R),\ (da \gg R)+1,\ (da \gg R)-2^{n-R},\ (da \gg R)-2^{n-R}+1\}$, where R is the RSH constant (TEA_RSH_CONST).

Parameters

k	bit position: WORD_SIZE $> k \geq 0$.
new_da	input difference to RSH partially constructed up to bit k.
new_dc	output difference from RSH partially constructed up to bit k.

Returns

TRUE if dc, after being fully constructed, will be a valid output difference from RSH, given the input difference da; FALSE otherwise.

Attention

The function is *not* optimal, meaning that it is overly-restrictive: all diferences (da, dc) which pass the checks are valid, but there also exist valid differences that do not pass the checks. The reason is that it is hard to detect all valid differences before they have been fully constructed.

More Details:

Given are two differences da and dc, that are only partially constructed up to bit k (counting from the LSB $k = 0$). rsh_condition_is_sat performs checks on da and dc and outputs if dc is such that $dc = da \gg R$, where R = TEA_RSH_CONST. The idea is to be able to discard pairs of diferences (da, dc) before they have been fully constructed. This allows to more efficiently constrct a list of valid differentials for the TEA F-function recursively. We use these conditions in tea_f_add_pddt_i to discard invalid entries early in the recursion.

To perform the checks, the following relations are used:

$dc = (da \gg R) \Longrightarrow dc \in \{dc_0, dc_1, dc_2, dc_3\}$ where:

- $dc_0 = (da \gg R)$.
- $dc_2 = (da \gg R) - 2^{n-R}$.
- $dc_1 = (da \gg R) + 1$.
- $dc_3 = (da \gg R) - 2^{n-R} + 1$.

Depending on the bit position k (some of) the following checks are performed:

1. If $(k \geq R)$ perform check on the $(k - R)$ LS bits. If $(k >= R)$ we check if the first $(k - R)$ LSB bits of $(da \gg R)$ are equal to the first $(k - R)$ bits of dc_i, $0 \leq i < 4$ according to the above equations. So we check if any of the following four equations hold:
   - $(da \gg R)[0 : (k - R)] = (dc_0)[0 : (k - R)]$.
   - $(da \gg R)[0 : (k - R)] = (dc_0 + 2^{n-R})[0 : (k - R)]$.
   - $(da \gg R)[0 : (k - R)] = (dc_0 - 1)[0 : (k - R)]$.
   - $(da \gg R)[0 : (k - R)] = (dc_0 + 2^{n-R} - 1)[0 : (k - R)]$.

2. Check that the R LS bits of da are not zero $da[(r-1):0] \neq 0$.
3. If $(k >= R) \wedge (k > (n-R))$ check the $(n-R)$ MS bits. When $(k > (n-R))$, $(da \gg R)[k] = 0$ and we check the top $(n-R)$ MS bits of dc. More specifically, we check if the initial four equations hold for the $(n-R)$ MS bits of the operands:
   - $dc_0[(n-1):(n-R+1)] = (da \gg R)[(n-1):(n-R+1)]$.
   - $dc_1[(n-1):(n-R+1)] = ((da \gg R)+1)[(n-1):(n-R+1)]$.
   - $dc_2[(n-1):(n-R+1)] = ((da \gg R)-2^{n-R})[(n-1):(n-R+1)]$.
   - $dc_3[(n-1):(n-R+1)] = ((da \gg R)-2^{n-R}+1)[(n-1):(n-R+1)]$.

7.61.2.3 void tea_f_add_pddt (uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::set< differential_t, struct_comp_diff_dx_dy > * *diff_set_dx_dy*)

Compute a partial DDT (pDDT) for the TEA F-function: wrapper function of tea_f_add_pddt_i . By definition a pDDT contains only differentials that have probability above a fixed probability thershold.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is TEA_ADD_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_set_dx_dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

tea_f_add_pddt_i.

7.61.2.4 void tea_f_add_pddt_adjust_to_key (uint32_t *nrounds*, uint32_t *npairs*, uint32_t *key[4]*, double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > * *diff_set_dx_dy*)

Adjust the probabailities of the differentials in a pDDT computed with tea_f_add_pddt , to the value of a fixed key by performing one-round TEA encryptions over a number of chosen plaintext pairs drawn uniformly at random.

Parameters

nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
key	cryptographic key of TEA.
p_thres	probability threshold (TEA_ADD_P_THRES).
diff_set_dx_dy	set of differentials (the pDDT) ordered by index $i = (dx\, 2^n + dy)$ - smallest first.

7.61.2.5 void tea_f_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > * *diff_mset_p*, const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

From a pDDT represented in the form of a set of differentials ordered by index, compute a pDDT as a set of differentials ordered by probability.

Parameters

diff_mset_p	output pDDT: set of differentials $(dx \rightarrow dy)$ ordered by probability; stored in an STL multiset structure, internally implemented as a Red-- Black binary search tree.
diff_set_dx_-dy	input pDDT: set of differentials $(dx \rightarrow dy)$ ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red-- Black binary search tree.

7.61.2.6 void tea_f_add_pddt_exper (gsl_matrix * *A[2][2][2][2]*, uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset< differential_t, struct_comp_diff_p > * *diff_mset_p*)

Experimentally compute the full DDT of the TEA F-function containining expected probabilities, averaged over all keys and round constants. An exhautive search is performed over all input and output differences. **Complexity:** $O(2^{2n})$.

Parameters

A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is TEA_ADD_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	set of differentials $(dx \rightarrow dy)$ ordered by probability (the DDT).

7.61.2.7 void tea_f_add_pddt_fk_exper (uint32_t *n*, double *p_thres*, uint32_t *delta*, uint32_t *k0*, uint32_t *k1*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset< differential_t, struct_comp_diff_p > * *diff_mset_p*)

Experimentally compute the full DDT of the TEA F-function containining probabilities for a fixed key and round constant. An exhautive search is performed over all input and output differences. **Complexity:** $O(2^{2n})$.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is TEA_ADD_P_THRES).
delta	round constant.
k0	first round key.
k1	second round key.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).

diff_mset_p	set of differentials $(dx \to dy)$ ordered by probability (the DDT).

7.61.2.8 void tea_f_add_pddt_i (const uint32_t *k*, const uint32_t *n*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, gsl_matrix $*$ *A[2][2][2][2]*, gsl_vector $*$ *C*, uint32_t $*$ *da*, uint32_t $*$ *db*, uint32_t $*$ *dc*, uint32_t $*$ *dd*, double $*$ *p*, const double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > $*$ *diff_set_dx_dy*)

Computes a partial difference distribution table (pDDT) for the F-function of block cipher TEA.

Parameters

k	current bit position in the recursion.
n	word size (default is WORD_SIZE).
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
A	transition probability matrices for $\mathrm{adp}^{3\oplus}$ (adp_xor3_sf).
C	unit column vector for computing $\mathrm{adp}^{3\oplus}$ (adp_xor3).
da	first input difference to the XOR operation in F.
db	second input difference to the XOR operation in F.
dc	third input difference to the XOR operation in F.
dd	output difference from the XOR operation in F.
p	probability of the partially constructed differential $(da[k:0], db[k:0], dc[k:0] \to dd[k:0])$.
p_thres	probability threshold (default is TEA_ADD_P_THRES).
diff_set_dx_-dy	set of differentials $(dx \to dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Attention

The computed pDDT is based on the expected additive differential probability of the TEA F-function (eadp_tea_f), averaged over all round keys and round constants δ and therefore contains average (as opposed to fixed-key fixed-constants adp_tea-_f_fk) probabilities.

Algorithm Outline:

Applies conceptually the same logic as adp_xor_pddt_i. It recursively constructs all differentials for the XOR operation with three inputs $(da, db, dc \to dd)$, with the additional requirement that they must satisfy the following properties:

1. $\mathrm{adp}^{3\oplus}(da, db, dc \to dd) > p_{\mathrm{thres}}$.
2. $db = da \ll 4$.
3. $dc \in (da \ll R), (da \ll R) + 1, (da \ll R) - 2^{n-R}, (da \ll R) - 2^{n-R} + 1$, so that $dc = (da \ll R)$ where $R =$ TEA_RSH_CONST.

Only the entries for which $\mathrm{eadp}^F(da \to dd) > p_{\mathrm{thres}}$ are stored.

See also

adp_xor_pddt_i, lsh_condition_is_sat, rsh_condition_is_sat.

7.62 src/tea.cc File Reference

Common functions used in the analysis of TEA.

`#include "common.hh" #include "tea.hh"`

Functions

- void tea_encrypt (uint32_t ∗v, uint32_t ∗k, int nrounds)
- uint32_t tea_f (uint32_t x, uint32_t k0, uint32_t k1, uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- uint32_t tea_f_i (const uint32_t mask_i, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t x_in)
- void tea_compute_delta_const (uint32_t D[TEA_NCYCLES])
- double **tea_add_diff_adjust_to_key** (const uint64_t npairs, const int round_idx, const uint32_t da, const uint32_t db, const uint32_t key[4])
- double **tea_differential_thres_exper_fk** (uint64_t npairs, int r, uint32_t key[4], uint32_t da[2], uint32_t db[2])
- uint32_t **tea_add_verify_trail** (uint32_t nrounds, uint32_t npairs, uint32_t key[4], differential_t trail[NROUNDS])
- uint32_t **tea_add_verify_differential** (uint32_t nrounds, uint32_t npairs, uint32_t key[4], differential_t trail[NROUNDS])
- void **print_trail_latex** (FILE ∗fp, uint32_t nrounds, uint32_t keys[4], differential_t trail[NROUNDS])

7.62.1 Detailed Description

Common functions used in the analysis of TEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.62.2 Function Documentation

7.62.2.1 void tea_compute_delta_const (uint32_t *D[TEA_NCYCLES]*)

Compute all round constants of block cipher TEA.

Parameters

D	all round constants δ of TEA.

7.62.2.2 void tea_encrypt (uint32_t * v, uint32_t * k, int nrounds)

Round-reduced version of block cipher TEA. Reference: https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm.

Parameters

v	plaintext.
k	secret key.
nrounds	number of rounds ($1 \leq$ `nrounds` ≤ 64).

7.62.2.3 uint32_t tea_f (uint32_t x, uint32_t k0, uint32_t k1, uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)

The F-function of block cipher TEA: $F(x) = ((x \ll 4) + k_0) \oplus (x + \delta) \oplus ((x \gg 5) + k_1)$.

Parameters

x	input to F.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$F(x)$

7.62.2.4 uint32_t tea_f_i (const uint32_t mask_i, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t x_in)

The F-function of block cipher TEA (tea_f) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	input to F.

Returns

$F(x) \bmod 2^i$

Attention

the initial value `x_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f()

7.63 src/xdp-add-pddt.cc File Reference

Compute a partial difference distribution table (pDDT) for xdp^{+}.

`#include "common.hh" #include "xdp-add.hh"`

Functions

- uint32_t xdp_add_pddt_exper (std::multiset< differential_3d_t, struct_comp_diff_3d_p > *diff_set, double p_thres)
- void xdp_add_pddt_i (const uint32_t k, const uint32_t n, const double p_thres, gsl_matrix *A[2][2][2], gsl_vector *C, uint32_t *da, uint32_t *db, uint32_t *dc, double *p, std::multiset< differential_3d_t, struct_comp_diff_3d_p > *diff_set)
- void xdp_add_pddt (uint32_t n, double p_thres)

7.63.1 Detailed Description

Compute a partial difference distribution table (pDDT) for xdp^{+}.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.63.2 Function Documentation

7.63.2.1 void xdp_add_pddt (uint32_t *n*, double *p_thres*)

Compute a partial DDT for xdp^{+}: wrapper function of xdp_add_pddt_i.

Parameters

n	word size.
p_thres	probability threshold.

See also

xdp_add_pddt_i.

7.63.2.2 uint32_t **xdp_add_pddt_exper** (std::multiset< **differential_3d_t**, **struct_comp_diff_3d_p** > ∗ *diff_set*, double *p_thres*)

Compute a partial DDT for xdp^{+} by exhasutive search over all input and output differences.

Parameters

diff_set	set of all differentials with probability not less than the threshold (the pDDT)
p_thres	probability threshold.

Returns

number of elements in the pDDT.

See also

xdp_add_pddt_i

7.63.2.3 void **xdp_add_pddt_i** (const uint32_t *k*, const uint32_t *n*, const double *p_thres*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, double ∗ *p*, std::multiset< **differential_3d_t**, **struct_comp_diff_3d_p** > ∗ *diff_set*)

Recursively compute all XOR differentials $(da, db \rightarrow dc)$ for ADD that have probability xdp^{+} larger than a fixed probability threshold `p_thres`.

The function works recursively starting from the LS bit `k` = 0 and terminating at the -MS bit `n`. At every bit position i it assigns values to the i-th bits of the differences `da`, `db`, `dc` and evaluates the probability of the resulting partial (i+1)-bit differential: $(da[i:0], db[i:0] \rightarrow dc[i:0])$. The recursion proceeds only if this probability is not less than the threshold `p_thres`. When i = n, the differential $(da[n-1:0], db[n-1:0] \rightarrow dc[n-1:0])$ is stored in an STL multiset structure (internally implemented as a Red-Black tree).

The **complexity** is strongly dependent on the threshold and is worst-case exponential in the word size: $O(2^{3n})$.

Note

If `p_thres` = 0.0 then the full DDT is computed.
Can be used also to compute all differentials that have non-zero probability by setting p_thres > 0.0 .
For 32 bit words, recommended values for the threshold are p_thres >= 0.7.

Parameters

k	current bit position in the recursion.
n	word size.
p_thres	probability threshold.
A	transition probability matrices for xdp^{+}.
da	first input difference.
db	second input difference.
dc	output difference.
p	probability of the differential $(da[k:0], db[k:0] \rightarrow dc[k:0])$.
diff_set	set of all differentials with probability not less than the threshold (the pDDT)

7.64 src/xdp-add-program.cc File Reference

Program for computing xdp^{+} with user-provided input.

`#include "common.hh" #include "xdp-add.hh"`

Functions

- void xdp_add_program ()
- int main ()

7.64.1 Detailed Description

Program for computing xdp^{+} with user-provided input.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.64.2 Function Documentation

7.64.2.1 int main ( )

Main function for the XDP-ADD program.

7.64.2.2 void xdp_add_program ( )

Compute XDP-ADD with user-provided input.

7.65 src/xdp-add.cc File Reference

The XOR differential probability of ADD $\mathrm{xdp}^{+}(da, db \rightarrow db)$.

`#include "common.hh" #include "xdp-add.hh"`

Functions

- void xdp_add_alloc_matrices (gsl_matrix *A[2][2][2])
- void xdp_add_free_matrices (gsl_matrix *A[2][2][2])
- void xdp_add_normalize_matrices (gsl_matrix *A[2][2][2])
- void xdp_add_print_matrices (gsl_matrix *A[2][2][2])
- void xdp_add_sf (gsl_matrix *A[2][2][2])
- double xdp_add (gsl_matrix *A[2][2][2], uint32_t da, uint32_t db, uint32_t dc)
- double xdp_add_exper (const uint32_t da, const uint32_t db, const uint32_t dc)

7.65.1 Detailed Description

The XOR differential probability of ADD $\mathrm{xdp}^{+}(da, db \rightarrow db)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.65.2 Function Documentation

7.65.2.1 double xdp_add (gsl_matrix * *A[2][2][2]*, uint32_t *da*, uint32_t *db*, uint32_t *dc*)

The XOR differential probability of ADD (xdp^{+}). **Complexity:** $O(n)$.

Parameters

A	transition probability matrices for xdp^{+} computed with xdp_add_sf.
da	first input difference.
db	second input difference.
dc	output difference.

Returns

$p = \mathrm{xdp}^{+}(da, db \rightarrow dc)$

See also

adp_xor

7.65.2.2 void xdp_add_alloc_matrices (gsl_matrix * *A[2][2][2]*)

Allocate memory for the transition probability matrices for xdp^{+}.

Parameters

A	transition probability matrices for xdp^{+}.

See also

xdp_add_free_matrices

7.65.2.3 double xdp_add_exper (const uint32_t *da*, const uint32_t *db*, const uint32_t *dc*)

The XOR differential probability of ADD (xdp^{+}) computed experimentally over all inputs. **Complexity:** $O(2^{2n})$.

Parameters

da	first input difference.
db	second input difference.
dc	output difference.

Returns

$p = \mathrm{xdp}^{+}(da, db \rightarrow dc)$

See also

xdp_add

7.65.2.4 void xdp_add_free_matrices (gsl_matrix * *A[2][2][2]*)

Free memory reserved by a previous call to xdp_add_alloc_matrices.

Parameters

A	transition probability matrices for xdp^{+}.

7.65.2.5 void xdp_add_normalize_matrices (gsl_matrix * *A[2][2][2]*)

Transform the elements of A into probabilities.

Parameters

A	transition probability matrices for xdp^+.

7.65.2.6 void xdp_add_print_matrices (gsl_matrix ∗ *A[2][2][2]*)

Print the matrices for xdp^+.

Parameters

A	transition probability matrices for xdp^+.

7.65.2.7 void xdp_add_sf (gsl_matrix ∗ *A[2][2][2]*)

S-function for xdp^+: $\mathrm{xdp}^+(da, db \rightarrow db)$.

Parameters

A	zero-initialized set of matrices.

Returns

Transition probability matrices A for $\mathrm{xdp}^+(da, db \rightarrow db)$.

$A[2][2][2] = A[da[i]][db[i]][dc[i]]$, where

- $da[i]$: the i-th bit of the first input difference.
- $db[i]$: the i-th bit of the second input difference.
- $dc[i]$: the i-th bit of the output difference.

See also

adp_xor_sf

7.66 src/xdp-tea-f-fk.cc File Reference

The XOR differential probability (XDP) of the F-function of TEA for a fixed key and round constants: $\mathrm{xdp}^F(k_0, k_1, \delta | \; da \rightarrow dd)$.

`#include "common.hh" #include "tea.hh"`

Functions

- double xdp_f_fk_exper (const uint32_t da, const uint32_t db, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- double max_xdp_f_fk_dx_exper (uint32_t ∗max_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)

- double max_xdp_f_fk_dy_exper (const uint32_t dx, uint32_t ∗max_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, uint32_t lsh_const, uint32-_t rsh_const)
- bool xdp_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32-_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, int32_t x)
- bool xdp_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- uint32_t xdp_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_-t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double xdp_f_fk (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- uint32_t xdp_f_assign_bit_x_dx (const uint32_t n, const uint32_t i, const uint32-_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dx)
- double max_dx_xdp_f_fk (const uint32_t n, uint32_t ∗ret_dx, const uint32_t dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_-const, const uint32_t rsh_const)
- uint32_t xdp_f_assign_bit_x_dy (const uint32_t n, const uint32_t i, const uint32-_t mask_i, const uint32_t x, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗ret_prob, uint32_t ∗ret_dy)
- double max_dy_xdp_f_fk (const uint32_t n, const uint32_t dx, uint32_t ∗ret_dy, const uint32_t k0, const uint32_t k1, const uint32_t delta, const uint32_t lsh_-const, const uint32_t rsh_const)

7.66.1 Detailed Description

The XOR differential probability (XDP) of the F-function of TEA for a fixed key and round constants: $\mathrm{xdp}^F(k_0, k_1, \delta | \, da \rightarrow dd)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Attention

The algorithms in this file have complexity that depends on the input and output differences to F. It is worst-case exponential in the word size, but is sub-exponential on average.

7.66.2 Function Documentation

7.66.2.1 double max_dx_xdp_f_fk (const uint32_t *n*, uint32_t * *ret_dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \to dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
ret_dx	maximum probability input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \to dy)$.

See also

xdp_f_assign_bit_x_dx

7.66.2.2 double max_dy_xdp_f_fk (const uint32_t *n*, const uint32_t *dx*, uint32_t * *ret_dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \to dy)$. **Complexity:** $O(2n) < c \leq O(2^{2n})$. **Memory requirement:** $4 \cdot 2^n$ Bytes.

Parameters

n	word size.
dx	input difference.
ret_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dy} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

xdp_f_assign_bit_x_dy, max_dy_xdp_f_fk

7.66.2.3 double max_xdp_f_fk_dx_exper (uint32_t * *max_dx,* const uint32_t *dy,* const uint32_t *k0,* const uint32_t *k1,* const uint32_t *delta,* uint32_t *lsh_const,* uint32_t *rsh_const*)

For given output difference `dy`, compute the maximum probability input differences `dx` over all input differences: $\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

max_dx	maximum probability input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

max_dx_xdp_f_fk

7.66.2.4 double max_xdp_f_fk_dy_exper (const uint32_t *dx,* uint32_t * *max_dy,* const uint32_t *k0,* const uint32_t *k1,* const uint32_t *delta,* uint32_t *lsh_const,* uint32_t *rsh_const*)

For given input difference `dx`, compute the maximum probability output difference `dy` over all output differences: $\max_{dy} \mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$ through exhaustive search over all input values and input differences. **Complexity:** $O(2^{2n})$.

Parameters

dx	input difference.
max_dy	maximum probability output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$.

See also

max_dy_xdp_f_fk

7.66.2.5 uint32_t **xdp_f_assign_bit_x** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the F-function of TEA is satisfied. The function operates by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i + rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key XOR probability of `F`: $\mathrm{xdp}^F(k_0, k_1, \delta | \; dx \rightarrow dy)$.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_f_fk

7.66.2.6 uint32_t **xdp_f_assign_bit_x_dx** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dx*)

For given output difference `dy`, compute all input differences `dx` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability input difference.

The function works by recursively assigning the bits of `x` and `dx` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_f_is_sat .

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dx`.
ret_prob	the maximum probability over all input differences $\max_{dx} \mathrm{xdp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dx	the input difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_f_assign_bit_x, max_dx_xdp_f_fk

7.66.2.7 uint32_t xdp_f_assign_bit_x_dy (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *ret_prob*, uint32_t * *ret_dy*)

For given input difference `dx`, compute all output differences `dy` and their probabilities, by counting all values `x` that satisfy the differential $(dx \rightarrow dy)$ for a fixed key and round constant. At the same time keeps track of the maximum probability output difference.

The function works by recursively assigning the bits of `x` and `dy` starting at bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.

mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i` + `rsh_const`).
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x_cnt	array of 2^n counters - each one keeps track of the number of values satisfying $(dx \rightarrow dy)$ for every `dy`.
ret_prob	the maximum probability over all output differences $\max_{dy} \mathrm{xdp}^F(k_0, k_1, \delta \mid dx \rightarrow dy)$.
ret_dy	the output difference that has maximum probability.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_f_assign_bit_x_dx

7.66.2.8 bool xdp_f_check_x (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value `x` satisfies the XOR differential $(dx \rightarrow dy)$ for the TEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k_0, k_1, \delta : dy = F(x \oplus dx) \oplus F(x)$.

7.66.2.9 double **xdp_f_fk** (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher TEA: $\mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

xdp_f_assign_bit_x

7.66.2.10 double **xdp_f_fk_exper** (const uint32_t *da*, const uint32_t *db*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher TEA: $\mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k0	first round key.
k1	second round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k_0, k_1, \delta | \, dx \rightarrow dy)$.

See also

xdp_f_fk

7.66.2.11 bool xdp_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k0*, const uint32_t *k1*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, int32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` is satisfied on the `i` LS bits of `x` i.e. check if $k_0, k_1, \delta : \; dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0])$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k0	first round key.
k1	second round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i + rsh_const`).

Returns

TRUE if $k_0, k_1, \delta : \; dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0)$.

7.67 src/xdp-xtea-f-fk.cc File Reference

The XOR differential probability (XDP) of the F-function of XTEA for a fixed key and round constants: $\mathrm{xdp}^F(k, \delta | \, da \rightarrow dd)$.

`#include "common.hh" #include "max-xdp-add.hh" #include "xtea.hh"`

Functions

- double xdp_xtea_f_fk_exper (const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)

- double xdp_xtea_f_fk_approx (const uint32_t ninputs, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- bool xdp_xtea_f_check_x (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- bool xdp_xtea_f_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dx, const uint32_t dy, const uint32_t x)
- uint32_t xdp_xtea_f_assign_bit_x (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t x, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dx, const uint32_t dy, uint32_t ∗x_cnt, double ∗prob)
- double xdp_xtea_f_fk (const uint32_t n, const uint32_t dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double xdp_xtea_f2_fk_exper (const uint32_t daa, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double xdp_xtea_f2_fk_approx (const uint32_t ninputs, const uint32_t daa, const uint32_t da, const uint32_t db, const uint32_t k, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- bool xdp_xtea_f2_check_x_xx (const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dxx, const uint32_t dx, const uint32_t dy, const uint32_t xx, const uint32_t x)
- bool xdp_xtea_f2_is_sat (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t k, const uint32_t delta, const uint32_t dxx, const uint32_t dx, const uint32_t dy, const uint32_t xx, const uint32_t x)
- uint32_t xdp_xtea_f2_assign_bit_x_xx (const uint32_t n, const uint32_t i, const uint32_t mask_i, const uint32_t xx, const uint32_t x, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t dxx, const uint32_t dx, const uint32_t dy, uint64_t ∗x_cnt, double ∗prob)
- double xdp_xtea_f2_fk (const uint32_t n, const uint32_t dxx, const uint32_t dx, const uint32_t dy, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const)
- double nz_xdp_xtea_f (gsl_matrix ∗A[2][2][2], const uint32_t dx, uint32_t ∗dy, uint32_t lsh_const, uint32_t rsh_const)

7.67.1 Detailed Description

The XOR differential probability (XDP) of the F-function of XTEA for a fixed key and round constants: $\mathrm{xdp}^F(k, \delta | \, da \rightarrow dd)$.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

Attention

The algorithms in this file have complexity that depends on the input and output differences to F. It is worst-case exponential in the word size, but is sub-exponential on average.

See also

adp-xtea-f-fk.cc

7.67.2 Function Documentation

7.67.2.1 double nz_xdp_xtea_f (gsl_matrix ∗ *A[2][2][2],* const uint32_t *dx,* uint32_t ∗ *dy,* uint32_t *lsh_const,* uint32_t *rsh_const*)

For the XTEA F-function (xtea_f), for fixed input difference dx, compute an output difference dy such that the differential $(dx \rightarrow dy)$ has non-zero probability.

Parameters

A	transition probability matrices for xdp^{+} (xdp_add_sf).
dx	input difference.
dy	output difference.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^{F}(k, \delta | \, dx \rightarrow dy)$

Algorithm sketch:

1. Compute the output XOR difference after xtea_f_lxr : $dx_{\mathrm{LXR}} = (((dx \ll 4) \oplus (dx \gg 5))$.
2. Compute the maximum probability output difference dy after the modular addition of xtea_f : $p_{\max} = \max_{dy} \mathrm{xdp}^{+}(dx, dx_{\mathrm{LXR}} \rightarrow dy)$ (see max_xdp_add).
3. Store dy and return $p_{\max}$.

Attention

In the computation of $\max_{dy} \mathrm{xdp}$ the inputs to the addition are implicitly assumed to be independent. Clearly they are not and so the returned probability is only an approximation.

7.67.2.2 uint32_t xdp_xtea_f2_assign_bit_x_xx (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *xx*, const uint32_t *x*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, uint64_t * *x_cnt*, double * *prob*)

Counts the number of values `xx` and `x` for which the XOR differential $(dxx, dx \rightarrow dy)$ of the XTEA F-function with two inputs $F'(xx, x) = xx + F(x)$ (see xtea_f2) is satisfied. The algorithm operates by recursively assigning the bits of `xx` and `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_xtea_f2_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
xx	first input value.
x	second input value of size at least (`i` + `rsh_const`).
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
dxx	first input difference.
dx	second input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key XOR probability of `F'` : $\mathrm{xdp}^{F'}(k, \delta \mid dx \rightarrow dy)$.

Returns

`1` if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; `0` otherwise.

See also

xdp_xtea_f2_fk

Note

x_cnt counts both the values for x and for xx.

7.67.2.3 bool xdp_xtea_f2_check_x_xx (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *xx*, const uint32_t *x*)

Check if given input values `xx` and `x` satisfy the XOR differential $(dxx, dx \rightarrow dy)$ of the XTEA F-function with two inputs $F'(xx, x) = xx + F(x)$ (see xtea_f2).

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dxx	first input difference.
dx	second input difference.
dy	output difference.
xx	first input value.
x	second input value.

Returns

TRUE if $k, \delta: \; dy = F'(xx \oplus dxx, \; x \oplus dx) \oplus F'(xx, \; x)$.

7.67.2.4 double xdp_xtea_f2_fk (const uint32_t *n*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of of the XTEA F-function with two inputs $F'(xx,x) = xx + F(x)$ (see xtea_f2): $\mathrm{xdp}^F(k, \delta | \; dxx, dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^{2n})$.

Parameters

n	word size.
dxx	first input difference.
dx	second input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^{F'}(k, \delta | \; dxx, dx \rightarrow dy)$.

See also

xdp_xtea_f2_assign_bit_x_xx

7.67.2.5 double xdp_xtea_f2_fk_approx (const uint32_t *ninputs*, const uint32_t *daa*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the XDP of the XTEA F-function with two inputs $F'(xx,x) = xx + F(x)$ (see xtea_f2), obtained over a number of input chosen plaintext pairs c hosen uniformly at random.

Parameters

ninputs	number of input chosen plaintext pairs.
daa	first input difference.
da	second input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^{F'}(k,\delta|\ daa,da \rightarrow dy)$.

7.67.2.6 double **xdp_xtea_f2_fk_exper** (const uint32_t *daa*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key through exhaustive search over all input values the fixed-constant XOR differential probability of the F-function of block cipher XTEA including the second modular addition and denoted by $F'(xx,x) = xx + F(x)$ (see xtea_f2): $\mathrm{xdp}^{F'}(k,\delta|\ dxx,dx \rightarrow dy)$. **Complexity:** $O(2^{2n})$.

Parameters

daa	first input difference.
da	second input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^{F'}(k,\delta|\ daa,da \rightarrow dy)$.

7.67.2.7 bool **xdp_xtea_f2_is_sat** (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dxx*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *xx*, const uint32_t *x*)

Check if the XOR differential $(dxx,dx \rightarrow dy)$ of the XTEA F-function with two inputs $F'(xx,x) = xx + F(x)$ (see xtea_f2) is satisfied on the `i` LS bits of `xx` and `x` i.e. check if

$$k,\delta:\ dy[i-1:0] = F(xx[i-1:0], x[i-1:0] \oplus dx[i-1:0]) \oplus F(xx[i-1:0], x[i-1:0])$$

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dxx	first input difference.
dx	second input difference.
dy	output difference.
xx	first input value.
x	second input value of size at least (`i + rsh_const`).

Returns

TRUE if the differential is satisfied; FALSE otherwise.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

7.67.2.8 uint32_t **xdp_xtea_f_assign_bit_x** (const uint32_t *n*, const uint32_t *i*, const uint32_t *mask_i*, const uint32_t *x*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *dx*, const uint32_t *dy*, uint32_t * *x_cnt*, double * *prob*)

Counts the number of values `x` for which the differential $(dx \rightarrow dy)$ for the F-function of XTEA is satisfied. The function operates by recursively assigning the bits of `x` starting from bit position `i` and terminating at the MS bit `n`. The recursion proceeds to bit $(i+1)$ only if the differential is satisfied on the `i` LS bits. This is checked by applying xdp_xtea_f_is_sat.

Parameters

n	word size (terminating bit popsition).
i	current bit position.
mask_i	mask on the `i` LS bits of `x`.
x	input value of size at least (`i + rsh_const`).
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.
dx	input difference.
dy	output difference.
x_cnt	number of values satisfying $(dx \rightarrow dy)$.
prob	the fixed-key XOR probability of `F`: $\mathrm{xdp}^F(k, \delta \mid dx \rightarrow dy)$.

Returns

1 if $x[i-1:0]$ satisfies $(dx[i-1:0] \rightarrow dy[i-1:0])$; 0 otherwise.

See also

xdp_xtea_f_fk

7.67.2.9 bool xdp_xtea_f_check_x (const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if a given value x satisfies the XOR differential $(dx \rightarrow dy)$ for the XTEA F-function.

Parameters

lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value.

Returns

TRUE if $k, \delta: \; dy = F(x \oplus dx) \oplus F(x)$.

7.67.2.10 double xdp_xtea_f_fk (const uint32_t *n*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher XTEA: $\mathrm{xdp}^F(k, \delta | \; dx \rightarrow dy)$. **Complexity:** $O(n) < c \leq O(2^n)$.

Parameters

n	word size.
dx	input difference.
dy	output difference.
key	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k, \delta | \; dx \rightarrow dy)$.

See also

xdp_xtea_f_assign_bit_x

7.67.2.11 double **xdp_xtea_f_fk_approx** (const uint32_t *ninputs*, const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

An approximation of the XDP of the XTEA F-function (xtea_f) obtained over a number of input chosen plaintext pairs chosen uniformly at random.

Parameters

ninputs	number of input chosen plaintext pairs.
da	input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\mathrm{xdp}^F(k, \delta|\ dx \rightarrow dy)$.

7.67.2.12 double **xdp_xtea_f_fk_exper** (const uint32_t *da*, const uint32_t *db*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*)

Compute the fixed-key, fixed-constant XOR differential probability of the F-function of block cipher XTEA: $\mathrm{xdp}^F(k, \delta|\ dx \rightarrow dy)$ through exhaustive search over all input values. **Complexity:** $O(2^n)$.

Parameters

da	input difference.
db	output difference.
k	round key.
delta	round constant.
lsh_const	LSH constant.
rsh_const	RSH constant.

Returns

$\text{xdp}^F(k, \delta | \ dx \rightarrow dy)$.

7.67.2.13 bool xdp_xtea_f_is_sat (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *k*, const uint32_t *delta*, const uint32_t *dx*, const uint32_t *dy*, const uint32_t *x*)

Check if the differential $(dx \rightarrow dy)$ for `F` (xtea_f) is satisfied on the `i` LS bits of `x` i.e. check if

$k, \delta : \ dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0])$.

Attention

`x` must be of size at least $(i+R)$ bits where `R` is the RSH constant of `F`.

Parameters

mask_i	`i` bit mask.
lsh_const	LSH constant.
rsh_const	RSH constant.
k	round key.
delta	round constant.
dx	input difference.
dy	output difference.
x	input value of size at least (`i + rsh_const`).

Returns

TRUE if $k, \delta : \ dy[i-1:0] = F(x[i-1:0] \oplus dx[i-1:0]) \oplus F(x[i-1:0])$.

7.68 src/xtea-add-threshold-search.cc File Reference

Automatic search for ADD differential trails in block cipher XTEA.

```
#include "common.hh" #include "adp-xor.hh" #include "max-adp-xor.-
hh" #include "adp-xor-fi.hh" #include "max-adp-xor-fi.-
hh" #include "adp-shift.hh" #include "xtea.hh" #include
"adp-xtea-f-fk.hh" #include "xtea-f-add-pddt.hh"
```

Functions

- void xtea_add_threshold_search (const int n, const int nrounds, const uint32_t npairs, const uint32_t round_key[64], const uint32_t round_delta[64], gsl_matrix ∗A[2][2][2], gsl_matrix ∗AA[2][2][2], double B[NROUNDS], double ∗Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS], uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p

$>$ *diff_mset_p, std::set$<$ differential_t, struct_comp_diff_dx_dy $>$ *diff_set_dx-_dy)
- void xtea_add_trail_search (uint32_t key[4], uint32_t round_key[64], uint32_-t round_delta[64])

7.68.1 Detailed Description

Automatic search for ADD differential trails in block cipher XTEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.68.2 Function Documentation

7.68.2.1 void xtea_add_threshold_search (const int *n*, const int *nrounds*, const uint32_t *npairs*, const uint32_t *round_key[64]*, const uint32_t *round_delta[64]*, gsl_matrix * *A[2][2][2]*, gsl_matrix * *AA[2][2][2]*, double *B[NROUNDS]*, double * *Bn*, const differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset$<$ differential_t, struct_comp_diff_p $>$ * *diff_mset_p*, std::set$<$ differential_t, struct_comp_diff_dx_dy $>$ * *diff_set_dx_dy*)

Automatic search for ADD differential trails in block cipher XTEA using pDDT.

Note

For more details on the algorithm see tea_add_threshold_search.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.
A	transition probability matrices for $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for XOR with fixed input $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ (adp_xor-_fixed_input_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best found probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best found differential trail for `nrounds`.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	set of differentials (dx, dy, p) (the pDDT) ordered by probability p.
diff_set_dx_dy	set of differentials (dx, dy, p) (the pDDT) ordered by index $i = (dx\, 2^n + dy)$.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

See also

tea_add_threshold_search.

7.68.2.2 void xtea_add_trail_search (uint32_t *key[4]*, uint32_t *round_key[64]*, uint32_t *round_delta[64]*)

Search for ADD differential trails in block cipher XTEA: wrapper function for tea_add_threshold_search.

Parameters

key	cryptographic key of XTEA.
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.

Algorithm Outline:

The procedure operates as follows:

1. Compute a pDDT for F (xtea_f_add_pddt).
2. Adjust the probabilities of the pDDT to the round key and constant (adp_xtea_f_approx).
3. Execute the search for differential trails for n rounds (n = NROUNDS) through a successive application of xtea_add_threshold_search :
   - Compute the best found probability on 1 round: $B[0]$.
   - Using $B[0]$ compute the best found probability on 2 rounds: $B[1]$.
   - $\ldots$
   - Using $B[0],\ldots,B[i-1]$ compute the best found probability on $(i+1)$ rounds: $B[i]$.
   - $\ldots$

    - Using $B[0],\ldots,B[n-2]$ compute the best found probability on n rounds: $B[n-1]$.

4. Print the best found trail on n rounds on standrad output and terminate.

See also

xtea_add_threshold_search, tea_add_trail_search

7.69 src/xtea-f-add-pddt.cc File Reference

Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher XTEA.

`#include "common.hh"` `#include "adp-xor.hh"` `#include "max-adp-xor.hh"` `#include "max-adp-xor-fi.hh"` `#include "adp-shift.hh"` × `#include "tea-f-add-pddt.hh"` `#include "xtea.hh"` `#include "adp-xtea-f-fk.hh"`

Functions

- void xtea_f_add_pddt_i (const uint32_t k, const uint32_t n, const uint32_t key, const uint32_t delta, const uint32_t lsh_const, const uint32_t rsh_const, gsl_matrix ∗A[2][2][2], gsl_matrix ∗AA[2][2][2], gsl_vector ∗C, uint32_t ∗da, uint32_t ∗db, uint32_t ∗dc, uint32_t ∗dd, double ∗p, const double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy)
- void xtea_f_add_pddt (uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, gsl_matrix ∗A[2][2][2], gsl_matrix ∗AA[2][2][2], gsl_vector ∗C, uint32_t key, uint32_t delta, std::set< differential_t, struct_comp_diff_dx_dy > ∗diff_set_dx_dy)
- void xtea_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > ∗diff_mset_p, const std::set< differential_t, struct_comp_diff_dx_dy > diff_set_dx_dy)

7.69.1 Detailed Description

Computing an ADD partial difference distribution table (pDDT) for the F-function of block cipher XTEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.69.2 Function Documentation

7.69.2.1 void xtea_add_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > * *diff_mset_p*, const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

From a pDDT represented in the from of a set of differentials ordered by index, compute a pDDT as a set of differentials ordered by probability.

Parameters

diff_mset_p	output pDDT: set of differentials $(dx \rightarrow dy)$ ordered by probability; stored in an STL multiset structure, internally implemented as a Red--Black binary search tree.
diff_set_dx_dy	input pDDT: set of differentials $(dx \rightarrow dy)$ ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

tea_f_add_pddt_dxy_to_dp

7.69.2.2 void xtea_f_add_pddt (uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, gsl_matrix * *A[2][2][2]*, gsl_matrix * *AA[2][2][2]*, gsl_vector * *C*, uint32_t *key*, uint32_t *delta*, std::set< differential_t, struct_comp_diff_dx_dy > * *diff_set_dx_dy*)

Compute a partial DDT (pDDT) for the XTEA F-function: wrapper function of xtea_f_add_pddt_i . By definition a pDDT contains only differentials that have probability above a fixed probability thershold.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is XTEA_ADD_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
A	transition probability matrices for $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for XOR with fixed input $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ (adp_xor_fixed_input_sf).
C	unit column vector for computing $\mathrm{adp}^{\oplus}$ (adp_xor).
key	round key.
delta	round constant.
diff_set_dx_dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

tea_f_add_pddt_i.

7.69.2.3 void xtea_f_add_pddt_i (const uint32_t *k*, const uint32_t *n*, const uint32_t *key*, const uint32_t *delta*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, gsl_matrix ∗ *A[2][2][2]*, gsl_matrix ∗ *AA[2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, uint32_t ∗ *dd*, double ∗ *p*, const double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy*)

Computes an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA.

Parameters

k	current bit position in the recursion.
n	word size (default is WORD_SIZE).
key	round key.
delta	round constant.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
A	transition probability matrices for XOR $\mathrm{adp}^{\oplus}$ (adp_xor_sf).
AA	transition probability matrices for XOR with fixed input $\mathrm{adp}^{\oplus}_{\mathrm{FI}}$ (adp_xor_fixed_input_sf).
C	unit column vector for computing $\mathrm{adp}^{\oplus}$ (adp_xor).
da	input difference to the F-function of XTEA.
db	output difference from the LSH operation in F.
dc	output difference from the RSH operation in F.
dd	output difference from the XOR operation in F.
p	probability of the partially constructed differential $(db[k:0], dc[k:0] \rightarrow dd[k:0])$ for the XOR operation in F.
p_thres	probability threshold (default is XTEA_ADD_P_THRES).
diff_set_dx_dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Algorithm Outline:

1. Recursively construct all differentials for the XOR operation in the f_{LXR} component of the F-function of XTEA (see xtea_f_lxr): $f_{\mathrm{LXR}}(a) = (((a \ll 4) \oplus (a \gg 5))$. Note that when doing this, we treat the two inputs $(a \ll 4)$ and $(a \gg 5)$ as independent inputs, denoted respectively by b and c. At every bit position in the recursion we require the corresponding partially constructed input differences da, db, dc and the output difference dd to satisfy conditions lsh_condition_is_sat and rsh_condition_is_sat. As a result, after the MSB is processed and $k = n$ the so constructed differences satisfy the following constions (see tea_f_add_pddt_i):
   (a) $\mathrm{adp}^{3\oplus}(db, dc \rightarrow dd) > p_{\mathrm{thres}}$.
   (b) $db = da \ll 4$.

(c) $dc \in (da \ll R), (da \ll R)+1, (da \ll R)-2^{n-R}, (da \ll R)-2^{n-R}+1$, so that $dc = (da \ll R)$ where $R =$TEA_RSH_CONST.

2. Set $dz = da + dd$ according to the feed-forward operation in F (see xtea_f) and compute the maximum probability output difference dy for the ADD operation with round key and δ (see xtea_f) with one fixed input: $\max \mathrm{adp}^{\oplus}_{\mathrm{FI}}((\mathrm{key}+\delta),\ dz \rightarrow dy)$.
3. Experimentally adjust the probability of the differential $\mathrm{adp}^{F}(da \rightarrow dy)$ to the full function F using adp_xtea_f_approx . Set the adjusted probability to $\hat{p}$.
4. Store $(da, dy, \hat{p})$ in the pDDT.

See also

tea_f_add_pddt_i

7.70 src/xtea-f-xor-pddt.cc File Reference

Computing an XOR partial difference distribution table (pDDT) for the F-function of block cipher XTEA.

`#include "common.hh"` `#include "xdp-add.hh"` `#include "xtea.hh"` `#include "xdp-xtea-f-fk.hh"`

Functions

- void xtea_f_xor_pddt_i (const uint32_t k, const uint32_t n, const uint32_t lsh_const, const uint32_t rsh_const, gsl_matrix *A[2][2][2], gsl_vector *C, uint32_t *da, uint32_t *db, uint32_t *dc, double *p, const double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void xtea_f_xor_pddt (uint32_t n, double p_thres, uint32_t lsh_const, uint32_t rsh_const, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void xtea_xor_pddt_adjust_to_key (uint32_t nrounds, uint32_t npairs, uint32_t lsh_const, uint32_t rsh_const, uint32_t key, uint32_t delta, double p_thres, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy)
- void xtea_xor_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p, const std::set< differential_t, struct_comp_diff_dx_dy > diff_set_dx_dy)

7.70.1 Detailed Description

Computing an XOR partial difference distribution table (pDDT) for the F-function of block cipher XTEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.70.2 Function Documentation

7.70.2.1 void xtea_f_xor_pddt (uint32_t *n*, double *p_thres*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy*)

Compute an XOR partial DDT (pDDT) for the XTEA F-function: wrapper function of xtea_f_xor_pddt_i . By definition a pDDT contains only differentials that have probability above a fixed probability thershold.

Parameters

n	word size (default is WORD_SIZE).
p_thres	probability threshold (default is XTEA_XOR_P_THRES).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_set_dx_dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Note

The compuation of the pDDT is based on the ADD operation in the XTEA F-function: the only non-linear componenet with respect to XOR differences.

See also

xtea_f_xor_pddt_i.

7.70.2.2 void xtea_f_xor_pddt_i (const uint32_t *k*, const uint32_t *n*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, gsl_matrix ∗ *A[2][2][2]*, gsl_vector ∗ *C*, uint32_t ∗ *da*, uint32_t ∗ *db*, uint32_t ∗ *dc*, double ∗ *p*, const double *p_thres*, std::set< differential_t, struct_comp_diff_dx_dy > ∗ *diff_set_dx_dy*)

Computes an ADD partial difference distribution table (pDDT) for the F-function of block cipher TEA.

Parameters

k	current bit position in the recursion.
n	word size (default is WORD_SIZE).
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
A	transition probability matrices for ADD xdp^{+} (xdp_add_sf).
C	unit column vector for computing xdp^{+} (xdp_add).
da	input difference to the F-function of XTEA.

db	output difference from the f_{LXR} component of F ((xtea_f_lxr)).
dc	output difference from the F-function of XTEA.
p	probability of the partially constructed differential $(da[k:0], db[k:0] \rightarrow dc[k:0])$ for the ADD operation in F.
p_thres	probability threshold (default is XTEA_XOR_P_THRES).
diff_set_dx_-dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

Algorithm Outline:

1. Treat the two inputs to the ADD operation: a and $b = ((a << 4)^(a >> 5))$ as independent.
2. Recursively construct a list of differentials $(da, db \rightarrow dc)$ for the ADD operation in F with probability bigger than p_{thres} (see xdp_add_pddt_i).
3. Of the constructed differentials store in an pDDT only those for which it holds $db = (da \ll 4) \oplus (da \gg 5)$.
4. Return pDDT.

See also

xtea_f_xor_pddt

7.70.2.3 void xtea_xor_pddt_adjust_to_key (uint32_t *nrounds*, uint32_t *npairs*, uint32_t *lsh_const*, uint32_t *rsh_const*, uint32_t *key*, uint32_t *delta*, double *p_thres*, std::set$<$ differential_t, struct_comp_diff_dx_dy $> *$ *diff_set_dx_dy*)

Adjust the probabailities of the differentials in a pDDT computed with xtea_f_xor_pddt , to the value of a fixed key by performing one-round TEA encryptions over a number of chosen plaintext pairs drawn uniformly at random.

Parameters

nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
key	round key.
delta	round constant.
p_thres	probability threshold (XTEA_XOR_P_THRES).
diff_set_dx_-dy	set of differentials $(dx \rightarrow dy)$ in the pDDT ordered by index $i = (dx\, 2^n + dy)$.

7.70.2.4 void xtea_xor_pddt_dxy_to_dp (std::multiset< differential_t, struct_comp_diff_p > * *diff_mset_p*, const std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy*)

From a pDDT represented in the from of a set of differentials ordered by index, compute a pDDT as a set of differentials ordered by probability.

Parameters

diff_mset_p	output pDDT: set of differentials $(dx \rightarrow dy)$ ordered by probability; stored in an STL multiset structure, internally implemented as a Red--Black binary search tree.
diff_set_dx_dy	input pDDT: set of differentials $(dx \rightarrow dy)$ ordered by index $i = (dx\,2^n + dy)$; stored in an STL set structure, internally implemented as a Red--Black binary search tree.

See also

xtea_add_pddt_dxy_to_dp

7.71 src/xtea-xor-threshold-search.cc File Reference

Automatic search for XOR differential trails in block cipher XTEA.

```
#include "common.hh" #include "xdp-add.hh" #include "max-xdp-add.-
hh" #include "xtea.hh" #include "xdp-xtea-f-fk.hh" #include
"xtea-f-xor-pddt.hh"
```

Functions

- double xtea_xor_init_estimate (uint32_t next_round, uint32_t lsh_const, uint32_t rsh_const, uint32_t npairs, gsl_matrix *A[2][2][2], double B[NROUNDS], differential_t trail[NROUNDS], std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy, uint32_t round_key[64], uint32_t round_delta[64])
- void xtea_xor_threshold_search (const int n, const int nrounds, const uint32_t npairs, const uint32_t round_key[64], const uint32_t round_delta[64], gsl_matrix *A[2][2][2], double B[NROUNDS], double *Bn, const differential_t diff_in[NROUNDS], differential_t trail[NROUNDS], uint32_t lsh_const, uint32_t rsh_const, std::multiset< differential_t, struct_comp_diff_p > *diff_mset_p, std::set< differential_t, struct_comp_diff_dx_dy > *diff_set_dx_dy, uint32_t dxx_init, uint32_t *dxx_init_in)
- void xtea_xor_trail_search (uint32_t key[4], uint32_t round_key[64], uint32_t round_delta[64])

7.71.1 Detailed Description

Automatic search for XOR differential trails in block cipher XTEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

7.71.2 Function Documentation

7.71.2.1 double xtea_xor_init_estimate (uint32_t *next_round*, uint32_t *lsh_const*, uint32_t *rsh_const*, uint32_t *npairs*, gsl_matrix $*$ *A[2][2][2]*, double *B[NROUNDS]*, differential_t *trail[NROUNDS]*, std::set$<$ differential_t, struct_comp_diff_dx_dy $>$ $*$ *diff_set_dx_dy*, uint32_t *round_key[64]*, uint32_t *round_delta[64]*)

Compute an initial estimate of the probability of a differential trail on $(n+1)$ rounds, by greedily extending the best found trail for n rounds.

Parameters

next_round	index of round $(n+1)$ to which a trail on n rounds will be extended.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
npairs	number of chosen plaintext pairs (NPAIRS).
A	transition probability matrices for xdp^{+} (xdp_add_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
trail	best found differential trail for n rounds.
diff_set_dx_dy	pDDT as a set of differentials (dx, dy, p) ordered by index $i = (dx\ 2^n + dy)$.
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.

See also

xtea_xor_trail_search

7.71.2.2 void xtea_xor_threshold_search (const int *n*, const int *nrounds*, const uint32_t *npairs*, const uint32_t *round_key[64]*, const uint32_t *round_delta[64]*, gsl_matrix $*$ *A[2][2][2]*, double *B[NROUNDS]*, double $*$ *Bn*, const differential_t *diff_in[NROUNDS]*, differential_t *trail[NROUNDS]*, uint32_t *lsh_const*, uint32_t *rsh_const*, std::multiset$<$ differential_t, struct_comp_diff_p $>$ $*$ *diff_mset_p*, std::set$<$ differential_t, struct_comp_diff_dx_dy $>$ $*$ *diff_set_dx_dy*, uint32_t *dxx_init*, uint32_t $*$ *dxx_init_in*)

Automatic search for XOR differential trails in block cipher TEA. using pDDT.

Parameters

n	index of the current round: $0 \leq n < \mathrm{nrounds}$.
nrounds	total number of rounds (NROUNDS).
npairs	number of chosen plaintext pairs (NPAIRS).
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.

A	transition probability matrices for xdp^{+} (xdp_add_sf).
B	array containing the best differential probabilities for i rounds: $0 \leq i < n$.
Bn	the best found probability on n rounds, updated dynamically.
diff_in	array of differentials.
trail	best found differential trail for `nrounds`.
lsh_const	LSH constant (TEA_LSH_CONST).
rsh_const	RSH constant (TEA_RSH_CONST).
diff_mset_p	pDDT as a set of differentials (dx, dy, p) ordered by probability p.
diff_set_dx_dy	pDDT as a set of differentials (dx, dy, p) ordered by index $i = (dx\ 2^n + dy)$.
dxx_init	initial left input difference to XTEA
dxx_init_in	the initial left input difference to XTEA corresponding to the best found trail (initialized to dxx_init and updated dynamically).

Attention

The pDDT contains differentials and their probabilities for the XTEA F-function F (xtea_f) as opposed to the function F' (xtea_f2) that also includes the second ADD operation. In other words, the pDDT does *not* take into account the differential probabilities arising from the second ADD operation. The latter are computed during the search.

The outline of the array of bounds B is the following:

- $B[0]$: best probability for 1 round.
- $B[1]$: best probability for 2 rounds.
- $\ldots$
- $B[i]$: best probability for $(i+1)$ rounds.
- $\ldots$
- $B[n-2]$: best probability for $(n-1)$ rounds.
- $B[n-1]$: best probability for n rounds.

More Details

The differential probability (DP) for one round of XTEA is computed as the product of the DP of F (xtea_f) and the DP of the modular addition in F' (xtea_f2). The functions F and F' are defined as: $F(x) = y = x + ((x \ll 4) \oplus (x \gg 5))$, $F'(xx, x) = yy = xx + (y \oplus (\delta + \mathrm{key}))$. Thus the DP of one round of XTEA is essentiallly the DP of F' and is approximated as:

$$\mathrm{xdp}^{F'}(dxx, dx \rightarrow dyy) = \mathrm{xdp}^{F}(dx \rightarrow dy) \cdot \mathrm{xdp}^{+}(dy, dxx \rightarrow dyy).$$

Attention

The pDDT contais entries of the form $(dx,\ dy,\ \mathrm{xdp}^F(dx \rightarrow dy))$. However, every entry in the arrays of differentials `trail` and `diff_in` contains elements of the form: $(dx,\ dyy,\ \mathrm{xdp}^{F'}(dxx, dx \rightarrow dyy))$. Although `trail` and `dif_in` do not contain the difference dxx, the latter can be easily computed noting that $dxx = dx_{-1}$, where dx_{-1} is the input difference to F from the previous round.

For more details on the search algorithm see tea_add_threshold_search .

See also

xtea_xor_trail_search

7.71.2.3 void xtea_xor_trail_search (uint32_t *key[4]*, uint32_t *round_key[64]*, uint32_t *round_delta[64]*)

Search for XOR differential trails in block cipher XTEA: wrapper function for tea_add_threshold_search.

Parameters

key	cryptographic key of XTEA.
round_key	all round keys for the full XTEA.
round_delta	all round constants for the full XTEA.

Algorithm Outline:

The procedure operates as follows:

1. Compute a pDDT for F (xtea_f_xor_pddt).
2. Execute the search for differential trails for n rounds (n = NROUNDS) through a successive application of xtea_xor_threshold_search :
   - Compute the best found probability on 1 round: $B[0]$.
   - Using $B[0]$ compute the best found probability on 2 rounds: $B[1]$.
   - ...
   - Using $B[0], \ldots, B[i-1]$ compute the best found probability on $(i+1)$ rounds: $B[i]$.
   - ...
   - Using $B[0], \ldots, B[n-2]$ compute the best found probability on n rounds: $B[n-1]$.
3. Print the best found trail on n rounds on standrad output and terminate.

See also

xtea_xor_threshold_search

7.72 src/xtea.cc File Reference

Common functions used in the analysis of block cipher XTEA.

`#include "common.hh" #include "xtea.hh"`

Functions

- void xtea_r (uint32_t nrounds, uint32_t v[2], uint32_t const k[4], uint32_t lsh_const, uint32_t rsh_const)
- uint32_t xtea_f (uint32_t x, uint32_t k, uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- uint32_t xtea_f_i (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t x_in, const uint32_t k, const uint32_t delta)
- uint32_t xtea_f2 (uint32_t xx, uint32_t x, uint32_t k, uint32_t delta, uint32_t lsh_const, uint32_t rsh_const)
- uint32_t xtea_f2_i (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t xx_in, const uint32_t x_in, const uint32_t k, const uint32_t delta)
- uint32_t xtea_f_lxr (uint32_t x, uint32_t lsh_const, uint32_t rsh_const)
- uint32_t xtea_f_lxr_i (const uint32_t mask_i, const uint32_t lsh_const, const uint32_t rsh_const, const uint32_t x_in)
- void xtea_all_round_keys_and_deltas (uint32_t key[4], uint32_t round_key[64], uint32_t round_delta[64])
- double **xtea_one_round_xor_differential_exper** (uint64_t npairs, int round_idx, uint32_t key, uint32_t delta, uint32_t daa, uint32_t da, uint32_t db)
- double **xtea_one_round_add_differential_exper** (uint64_t npairs, int round_idx, uint32_t key, uint32_t delta, uint32_t da, uint32_t db)
- double **xtea_xor_differential_exper_v2** (uint64_t npairs, int r, uint32_t key[4], uint32_t da[2], uint32_t db[2], uint32_t lsh_const, uint32_t rsh_const)
- double **xtea_add_differential_exper_v2** (uint64_t npairs, int r, uint32_t key[4], uint32_t da[2], uint32_t db[2], uint32_t lsh_const, uint32_t rsh_const)
- uint32_t **xtea_xor_verify_differential** (uint32_t nrounds, uint32_t npairs, uint32_t lsh_const, uint32_t rsh_const, uint32_t key[4], uint32_t dxx_init, differential_t trail[NROUNDS])
- uint32_t **xtea_add_verify_differential** (uint32_t nrounds, uint32_t npairs, uint32_t lsh_const, uint32_t rsh_const, uint32_t key[4], differential_t trail[NROUNDS])
- uint32_t **xtea_xor_verify_trail** (uint32_t nrounds, uint32_t npairs, uint32_t round_key[64], uint32_t round_delta[64], uint32_t dxx_init, differential_t trail[NROUNDS])
- uint32_t **xtea_add_verify_trail** (uint32_t nrounds, uint32_t npairs, uint32_t round_key[64], uint32_t round_delta[64], differential_t trail[NROUNDS])

7.72.1 Detailed Description

Common functions used in the analysis of block cipher XTEA.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.72.2 Function Documentation

7.72.2.1 void xtea_all_round_keys_and_deltas (uint32_t *key[4]*, uint32_t *round_key[64]*, uint32_t *round_delta[64]*)

Compute all round keys and round constants of block cipher XTEA.

Parameters

key	initial key.
round_key	all round keys.
round_delta	all round constants δ of XTEA.

7.72.2.2 uint32_t xtea_f (uint32_t *x*, uint32_t *k*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

The F-function of block cipher XTEA: $F(x) = ((((x \ll 4) \oplus (x \gg 5)) + x) \oplus (k + \delta)$.

Parameters

x	input to F.
k	round key.
delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$F(x)$

7.72.2.3 uint32_t xtea_f2 (uint32_t *xx*, uint32_t *x*, uint32_t *k*, uint32_t *delta*, uint32_t *lsh_const*, uint32_t *rsh_const*)

The F-function of block cipher XTEA including the modular addition with the input to the previous Fesitel round. It is denoted by F' and is defined as:

$F'(xx, x) = xx + F(x)$,

where $F(x)$ is the XTEA F-function (xtea_f).

Parameters

x	first input to F'.
xx	second input to F'.
k	round key.
delta	round constant.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$F(x,xx)$

7.72.2.4 uint32_t xtea_f2_i (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *xx_in*, const uint32_t *x_in*, const uint32_t *k*, const uint32_t *delta*)

The F'-function of block cipher XTEA (xtea_f2) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	first input to F'.
xx_in	second input to F'.
k	round key.
delta	round constant.

Returns

$F'(x,xx) \bmod 2^i$

Attention

the initial values `x_in` and `xx_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f_i()

7.72.2.5 uint32_t xtea_f_i (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *x_in*, const uint32_t *k*, const uint32_t *delta*)

The F-function of block cipher XTEA (xtea_f) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	input to F.
k	round key.
delta	round constant.

Returns

$F(x) \bmod 2^i$

Attention

the initial value `x_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f_lxr_i()

7.72.2.6 uint32_t xtea_f_lxr (uint32_t *x*, uint32_t *lsh_const*, uint32_t *rsh_const*)

This function represents a sub-component of the XTEA F-function denoted by f_{LXR} and defined as: $f_{\mathrm{LXR}}(x) = (((x \ll 4) \oplus (x \gg 5))$.

Note

With f_{LXR}, the F-function of XTEA (xtea_f) is expressed as: $F(x) = (f_{\mathrm{LXR}}(x) + x) \oplus (k + \delta)$.

Parameters

x	input to f_{LXR}.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

Returns

$f_{\mathrm{LXR}}(x)$

7.72.2.7 uint32_t xtea_f_lxr_i (const uint32_t *mask_i*, const uint32_t *lsh_const*, const uint32_t *rsh_const*, const uint32_t *x_in*)

The component f_{LXR} of the XTEA F-function (xtea_f_lxr) computed on the first `i` least-significant (LS) bits.

Parameters

mask_i	`i` bit LSB mask.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).
x_in	first input to f_{LXR}.

Returns

$f_{\mathrm{LXR}}(x) \bmod 2^i$

Attention

the initial value `x_in` must be minimum (`rsh_const` + 1) bits long so that it can be shifted right by `rsh_const` positions.

See also

xtea_f_i()

7.72.2.8 void xtea_r (uint32_t *nrounds*, uint32_t *v[2]*, uint32_t const *k[4]*, uint32_t *lsh_const*, uint32_t *rsh_const*)

Round-reduced version of block cipher XTEA. Reference: `https://en.wikipedia.org/wiki/XTEA`.

Parameters

nrounds	number of rounds ($1 \leq$ `nrounds` ≤ 64).
v	plaintext.
k	secret key.
lsh_const	LSH constant (default is 4).
rsh_const	RSH constant (default is 5).

7.73 tests/adp-rsh-xor-tests.cc File Reference

Tests for adp-rsh-xor.cc.

```
#include "common.hh"   #include "adp-shift.hh"   #include "adp-rsh-xor.hh"
```

Functions

- void **test_adp_rsh_xor_alloc** ()
- void **test_adp_rsh_xor_sf** ()
- void **test_adp_rsh_xor** ()
- int main ()

7.73.1 Detailed Description

Tests for adp-rsh-xor.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.73.2 Function Documentation

7.73.2.1 int main ( )

Main function of ADP-RSH-XOR tests.

7.74 tests/adp-shift-tests.cc File Reference

Tests for adp-shift.cc.

`#include "common.hh" #include "adp-shift.hh"`

Functions

- void **test_adp_lsh** ()
- void **test_adp_lsh_all** ()
- void **test_adp_rsh** ()
- void **test_adp_rsh_all** ()
- int **main** ()

7.74.1 Detailed Description

Tests for adp-shift.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.75 tests/adp-tea-f-fk-ddt-tests.cc File Reference

Tests for adp-tea-f-fk-ddt.cc.

`#include "common.hh" #include "tea.hh" #include "adp-tea-f-fk-ddt.-hh"`

Data Structures

- struct skey_t

Functions

- void **test_ddt** ()
- void **test_adp_f_ddt_vs_exper** ()
- void **test_max_adp_f_ddt_exper** ()
- void **test_max_adp_f_ddt_vs_exper_all** ()
- void **test_max_adp_f_rsddt_vs_exper_all** ()
- bool **operator**< (skey_t x, skey_t y)
- void **test_max_adp_f_ddt_wrt_keys** ()
- void **test_adp_f_ddt** ()
- int **main** ()

7.75.1 Detailed Description

Tests for adp-tea-f-fk-ddt.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013 Note: Infeasible for large word sizes (> 10 bits). Used only for tests and verification.

7.76 tests/adp-tea-f-fk-noshift-tests.cc File Reference

Tests for adp-tea-f-fk-noshift.cc.

`#include "common.hh" #include "tea.hh" #include "adp-tea-f-fk-noshift.-hh"`

Functions

- void **test_adp_f_op_noshift** ()
- void **test_adp_f_op_noshift_vs_exper_all** ()
- void **test_adp_f_op_noshift_sf** ()
- void **test_adp_f_op_noshift_exper** ()
- int **main** ()

7.76.1 Detailed Description

Tests for adp-tea-f-fk-noshift.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.77 tests/adp-tea-f-fk-tests.cc File Reference

Tests for adp-tea-f-fk.cc.

```
#include "common.hh" #include "tea.hh" #include "adp-tea-f-fk.-
hh"
```

Functions

- void **test_adp_f_fk_v2_vs_adp_f_fk_exper** ()
- void **test_adp_f_fk_v2_vs_adp_f_fk_exper_all** ()
- void **test_adp_f_fk** ()
- void **test_max_dx_adp_f_fk** ()
- void **test_max_key_dx_adp_f_fk** ()
- void **test_max_dx_dy_adp_f_fk** ()
- void **test_max_adp_f_fk_dx_vs_exper_all** ()
- void **test_adp_f_fk_vs_exper** ()
- void **test_adp_f_fk_vs_exper_all** ()
- void **test_adp_f_fk_vs_exper_all_shconst_and_diffs** ()
- void **test_adp_f_fk_vs_exper_all_shconst** ()
- void **test_max_dy_adp_f_fk** ()
- void **test_all_dy_adp_f_fk** ()
- void **test_max_dy_adp_f_fk_vs_exper_all** ()
- void **test_max_dx_dy_adp_f_fk_vs_exper_all** ()
- int **main** ()

7.77.1 Detailed Description

Tests for adp-tea-f-fk.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.78 tests/adp-xor-fi-tests.cc File Reference

Tests for adp-xor-fi.cc.

`#include "common.hh" #include "adp-xor-fi.hh"`

Functions

- void **test_adp_xor_fixed_input** ()
- void **test_adp_xor_fixed_input_all** ()
- int **main** ()

7.78.1 Detailed Description

Tests for adp-xor-fi.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.79 tests/adp-xor-pddt-tests.cc File Reference

Tests for adp-xor-pddt.cc.

`#include "common.hh" #include "adp-xor.hh" #include "adp-xor-pddt.hh"`

Functions

- void **test_adp_xor_ddt** ()
- int main ()

7.79.1 Detailed Description

Tests for adp-xor-pddt.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.79.2 Function Documentation

7.79.2.1 int main ( )

Main function of ADP-XOR-PDDT tests.

7.80 tests/adp-xor-tests.cc File Reference

Tests for adp-xor.cc.

`#include "common.hh" #include "adp-xor.hh"`

Functions

- void test_adp_xor_matrices ()
- void test_adp_xor_all ()
- void test_adp_xor ()
- int main ()

7.80.1 Detailed Description

Tests for adp-xor.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.80.2 Function Documentation

7.80.2.1 int main ( )

Main function of ADP-XOR tests.

7.80.2.2 void test_adp_xor ( )

Test ADP-XOR for random input and output ADD differences.

7.80.2.3 void test_adp_xor_all ( )

Compare ADP-XOR to the experimental value.

7.80.2.4 void test_adp_xor_matrices ( )

Test allocation and free of ADP-XOR matrices.

7.81 tests/adp-xor3-tests.cc File Reference

Tests for adp-xor3.cc.

`#include "common.hh" #include "adp-xor3.hh"`

Functions

- void **test_adp_xor3_sf** ()
- void **test_adp_xor3_alloc_matrices** ()
- void **test_adp_xor3** ()
- void **test_adp_xor3_all** ()
- int main ()

7.81.1 Detailed Description

Tests for adp-xor3.cc.

Author

V.Velichkov, `vesselin.velichkov@uni.lu`

Date

2012-2013

7.81.2 Function Documentation

7.81.2.1 int main ( )

Main function of ADP-XOR3 tests.

7.82 tests/adp-xtea-f-fk-tests.cc File Reference

Tests for adp-xtea-f-fk.cc.

```
#include "common.hh" #include "adp-xor.hh" #include "max-adp-xor.-
hh"  #include "adp-xor-fi.hh"  #include "max-adp-xor-fi.-
hh"  #include "adp-shift.hh"  #include "xtea.hh"  #include
"adp-xtea-f-fk.hh"
```

Functions

- void **test_adp_xtea_f_lxr** ()
- void **test_adp_xtea_f_lxr_all** ()
- void **test_adp_xtea_f_lxr_vs_exper_all** ()
- void **test_adp_xtea_f_approx** ()
- void **test_adp_xtea_f_approx_all** ()
- void **test_adp_xtea_f** ()
- void **test_adp_xtea_f_all** ()
- void **test_max_dy_adp_xtea_f** ()
- void **test_max_dx_adp_xtea_f** ()
- void **test_max_dy_adp_xtea_f_is_max** ()
- void **test_max_dx_adp_xtea_f_is_max** ()
- void **test_max_dy_adp_xtea_f_is_max_all** ()
- void **test_max_dx_adp_xtea_f_is_max_all** ()
- double **test_first_nz_adp_xtea_f** ()
- void **test_first_nz_adp_xtea_f_all** ()
- void **test_first_nz_adp_xtea_f_random** ()
- int **main** ()

7.82.1 Detailed Description

Tests for adp-xtea-f-fk.cc.

Author

V.Velichkov, `vesselin.velichkov@uni.lu`

Date

2012-2013

7.83 tests/eadp-tea-f-tests.cc File Reference

Tests for eadp-tea-f.cc.

```
#include "common.hh"  #include "adp-shift.hh"  #include
"adp-xor3.hh"  #include "max-adp-xor3-set.hh"  #include
"tea.hh" #include "eadp-tea-f.hh"
```

Functions

- void **test_adp_xor3_vs_eadp_tea_f** ()
- void **test_eadp_tea_f** ()
- void **test_eadp_tea_f_vs_exper_all** ()
- void **test_max_eadp_tea_f_is_max** ()
- int main ()

7.83.1 Detailed Description

Tests for eadp-tea-f.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.83.2 Function Documentation

7.83.2.1 int main ( )

Main function of EADP-TEA-F tests.

7.84 tests/max-adp-xor-fi-tests.cc File Reference

Tests for max-adp-xor-fi.cc.

```
#include "common.hh"   #include "adp-xor-fi.hh"   #include
"max-adp-xor-fi.hh"
```

Functions

- void **test_max_adp_xor_fixed_input** ()
- void **test_max_adp_xor_fixed_input_is_max** ()
- int **main** ()

7.84.1 Detailed Description

Tests for max-adp-xor-fi.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.85 tests/max-adp-xor-tests.cc File Reference

Tests for max-adp-xor.cc.

`#include "common.hh" #include "adp-xor.hh" #include "max-adp-xor.-hh"`

Functions

- void **test_max_adp_xor** ()
- void **test_max_adp_xor_is_max** ()
- int **main** ()

7.85.1 Detailed Description

Tests for max-adp-xor.cc.

Author

V.Velichkov, `vesselin.velichkov@uni.lu`

Date

2012-2013

7.86 tests/max-adp-xor3-set-tests.cc File Reference

Tests for max-adp-xor3-set.cc.

`#include "common.hh" #include "adp-xor3.hh" #include "max-adp-xor3-set.-hh"`

Functions

- void **test_max_adp_xor3_set** ()
- void **test_max_adp_xor3_set_is_max_all** ()
- void **test_max_adp_xor3_set_is_max_rand** ()
- int **main** ()

7.86.1 Detailed Description

Tests for max-adp-xor3-set.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.87 tests/max-adp-xor3-tests.cc File Reference

Tests for max-adp-xor3.cc.

`#include "common.hh"` `#include "adp-xor3.hh"` `#include "max-adp-xor3.hh"`

Functions

- void **test_max_adp_xor3_rec** ()
- void **test_max_adp_xor3_vs_rec_all** ()
- void **test_max_adp_xor3_is_max** ()
- int **main** ()

7.87.1 Detailed Description

Tests for max-adp-xor3.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.88 tests/max-xdp-add-tests.cc File Reference

Tests for max-xdp-add.cc.

`#include "common.hh"` `#include "xdp-add.hh"` `#include "max-xdp-add.hh"`

Functions

- void **test_max_xdp_add** ()
- void **test_max_xdp_add_is_max** ()
- int main ()

7.88.1 Detailed Description

Tests for max-xdp-add.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.88.2 Function Documentation

7.88.2.1 int main ( )

Main function of MAX-XDP-ADD tests.

7.89 tests/tea-add-ddt-search-tests.cc File Reference

Tests for tea-add-ddt-search.cc.

`#include "common.hh" #include "tea.hh" #include "adp-tea-f-fk-ddt.-hh" #include "tea-add-ddt-search.hh"`

Functions

- void **test_tea_search_ddt** ()
- void **test_tea_search_xddt** ()
- void **test_tea_search_xddt_bottom_up** ()
- void **test_tea_search_ddt_xddt_xddt_bottom_up** ()
- int **main** ()

7.89.1 Detailed Description

Tests for tea-add-ddt-search.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.90 tests/tea-add-threshold-search-tests.cc File Reference

Tests for tea-add-threshold-search.cc.

`#include "common.hh" #include "tea.hh" #include "tea-add-threshold-search.hh"`

Functions

- void **test_tea_add_trail_search** ()
- int **main** ()

7.90.1 Detailed Description

Tests for tea-add-threshold-search.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.91 tests/tea-f-add-pddt-tests.cc File Reference

Tests for tea-f-add-pddt.cc.

`#include "common.hh" #include "adp-xor3.hh" #include "adp-shift.hh" #include "tea.hh" #include "eadp-tea-f.hh" #include "adp-tea-f-fk.hh" #include "tea-f-add-pddt.hh"`

Functions

- void **test_rsh_condition** ()
- void **test_tea_f_add_pddt_vs_full_ddt** ()
- int **main** ()

7.91.1 Detailed Description

Tests for tea-f-add-pddt.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu Testing the computation of pDDT for the TEA F-function

7.92 tests/xdp-add-tests.cc File Reference

Tests for xdp-add.cc.

`#include "common.hh" #include "xdp-add.hh"`

Functions

- void test_xdp_add_matrices ()
- void test_xdp_add ()
- void test_xdp_add_all ()
- int main ()

7.92.1 Detailed Description

Tests for xdp-add.cc. Tests for xdp^{+}.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.92.2 Function Documentation

7.92.2.1 int main ( )

Main function of XDP-ADD tests.

7.92.2.2 void test_xdp_add ( )

Test XDP-ADD for random input and output XOR differences.

7.92.2.3 void test_xdp_add_all ( )

Compare XDP-ADD to the experimental value.

7.92.2.4 void test_xdp_add_matrices ( )

Test allocation and free of XDP-ADD matrices.

7.93 tests/xdp-tea-f-fk-tests.cc File Reference

Tests for xdp-tea-f-fk.cc.

`#include "common.hh" #include "tea.hh" #include "xdp-tea-f-fk.hh"`

Functions

- void **test_xdp_f_fk** ()
- void **test_xdp_f_fk_vs_exper_all** ()
- void **test_max_dx_xdp_f_fk** ()
- void **test_max_dx_xdp_f_fk_vs_exper_all** ()
- void **test_max_dx_xdp_f_fk_vs_exper** ()
- void **test_max_dy_xdp_f_fk** ()
- void **test_max_dy_xdp_f_fk_vs_exper_all** ()
- int **main** ()

7.93.1 Detailed Description

Tests for xdp-tea-f-fk.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.94 tests/xdp-xtea-f-fk-tests.cc File Reference

Tests for xdp-xtea-f-fk.cc.

`#include "common.hh" #include "xdp-add.hh" #include "xtea.hh" #include "xdp-xtea-f-fk.hh"`

Functions

- void **test_xdp_xtea_f_fk** ()
- void **test_xdp_xtea_f_fk_all** ()
- void **test_xdp_xtea_f2_fk** ()
- void **test_xdp_xtea_f2_fk_all** ()
- void **test_xdp_xtea_f2_fk_approx** ()
- void **test_nz_xdp_xtea_f** ()
- int **main** ()

7.94.1 Detailed Description

Tests for xdp-xtea-f-fk.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.95 tests/xtea-add-threshold-search-tests.cc File Reference

Tests for xtea-add-threshold-search.cc.

`#include "common.hh" #include "xtea.hh" #include "xtea-add-threshold-search.hh"`

Functions

- void **test_xtea_add_trail_search** ()
- int **main** ()

7.95.1 Detailed Description

Tests for xtea-add-threshold-search.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013

7.96 tests/xtea-xor-threshold-search-tests.cc File Reference

Tests for xtea-xor-threshold-search.cc.

`#include "common.hh" #include "xtea.hh" #include "xtea-xor-threshold-search.hh"`

Functions

- void **test_xtea_xor_trail_search** ()
- int **main** ()

7.96.1 Detailed Description

Tests for xtea-xor-threshold-search.cc.

Author

V.Velichkov, vesselin.velichkov@uni.lu

Date

2012-2013