The CodingTool Library

Tomislav Nad

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a, A-8010 Graz, Austria

Tomislav.Nad@iaik.tugraz.at

Outline

Introduction

CodingTool Library?
A cryptanalysis tool based on coding theory

Cryptanalysis based on Coding Theory

- Coding theory is a powerful tool in the cryptanalysis of cryptographic primitives
- Can be used to find differential characteristics with low Hamming weight for hash functions
- All differential characteristics for a linearized hash function can be seen as the code words of a linear code[7]

Linear Codes

- A linear code C is a subspace of a vector space over a finite field

$$C \leq \mathbb{F}_q^n$$

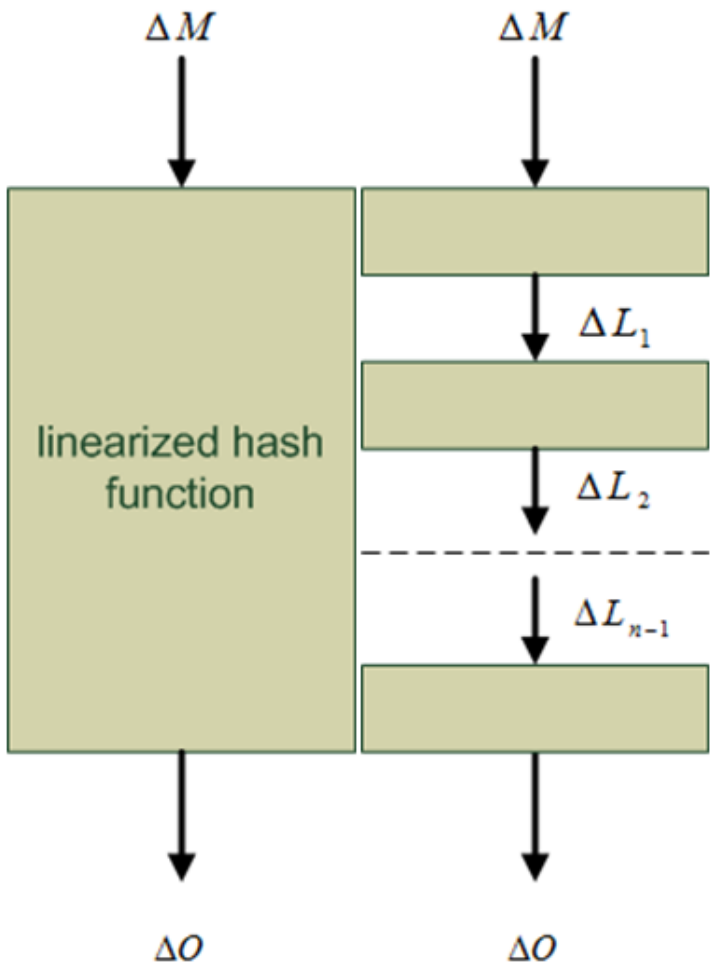
- Generator matrix (k linear independent code words)

$$G = (g_0, \ldots, g_k)^T$$

$$c = x \cdot G \text{ for } x \in \mathbb{F}_q^k$$

Hash Functions and Linear Codes

- A linearized hash function describes a linear code
- A code word describes a linear differential characteristic
- A code word with low Hamming weight represents a characteristic with few differences
- Compute *k* linear independent code words to form the generator matrix



Low Hamming Weight Search

- Searching for code words with low Hamming weight
- The fewer differences the higher the success probability of a differential
- Using a probabilistic algorithm
- Algorithm works with generator matrix

Many Results

- SHA-0 [3]
- SHA-1 [6]
- EnRUPT [4]
- CubeHash [1]
- SIMD [5]
- . . .

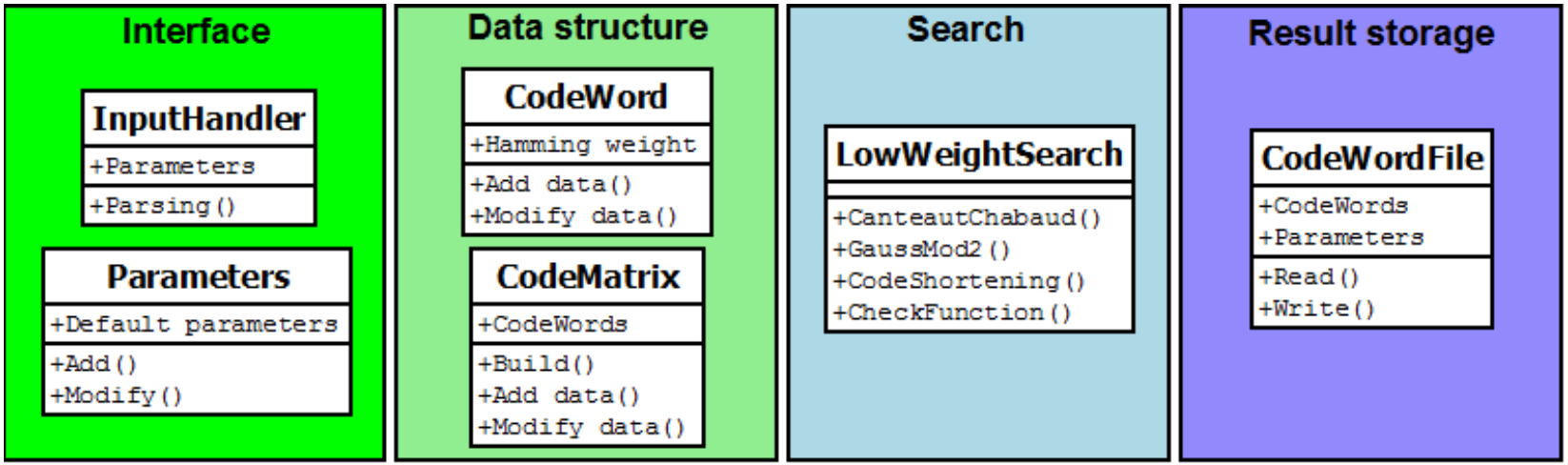
The CodingTool Library

- Automatic tool for cryptanalysis
- Implementation of search algorithm (Canteaut and Chabaud [2])
- High abstraction level
- Easy to use interface
- Modularity and extensibility

Overview (1/2)

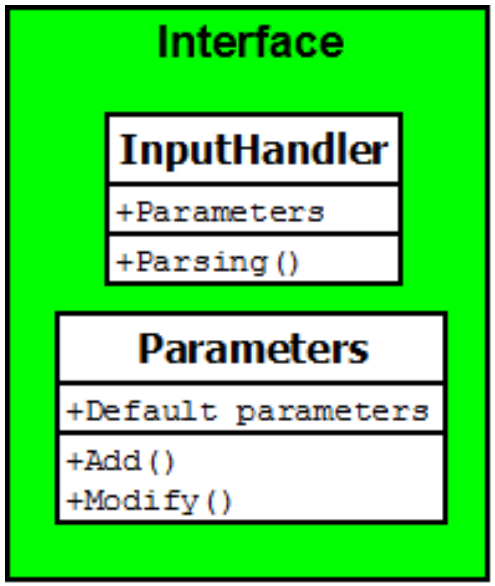
- Written in C++
- Depends only on the STL
- Multiplatform (Windows, Linux)
- Utilize 64-bit architecture
- Complete documentation and examples
- Licensed under GPL 3.0

Overview (2/2)


Interface
InputHandler
+Parameters
+Parsing()
Parameters
+Default parameters
+Add()
+Modify()
Data structure
CodeWord
+Hamming weight
+Add data()
+Modify data()
CodeMatrix
+CodeWords
+Build()
+Add data()
+Modify data()
Search
LowWeightSearch
+CanteautChabaud()
+GaussMod2()
+CodeShortening()
+CheckFunction()
Result storage
CodeWordFile
+CodeWords
+Parameters
+Read()
+Write()

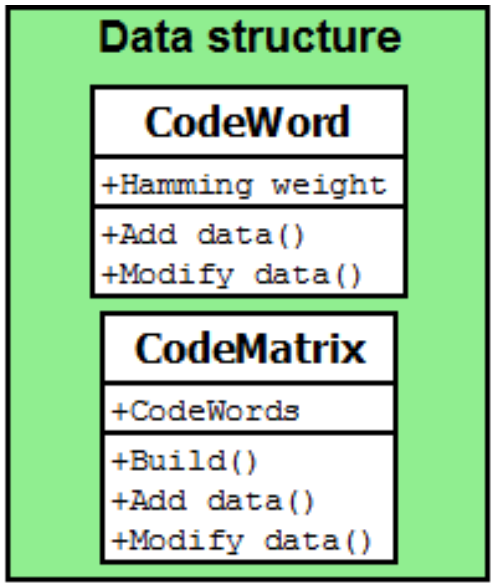
Interface

- Command line parser
- Set of default parameters
- Custom parameters



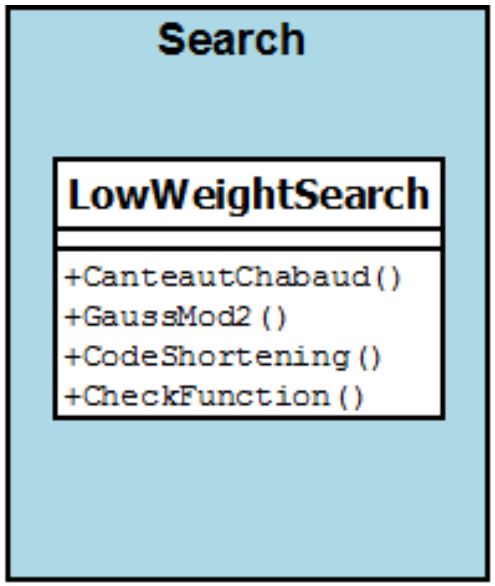
Data Structure

- Using maximum available word size
- User do not have to care about used word size
- User can add/modify data of different size (PushBool, Pop32, Erase64, . . .)



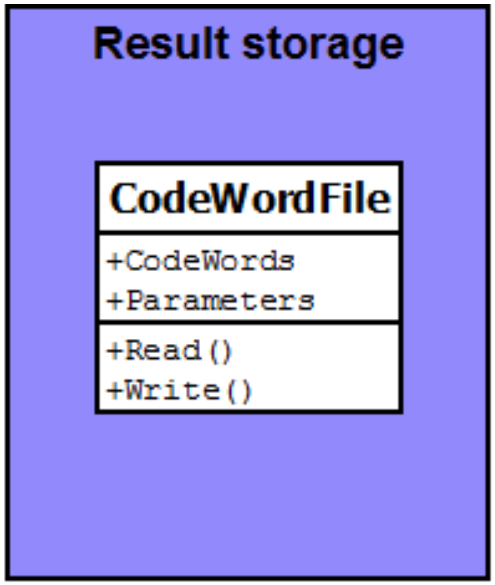
Search

- Code shortening: useful to find collision producing code words
- Check function: applied on each code word during the search



Result Storage

- Stores all important information in a file (code words, parameters, . . .)
- Can be easily reused



Example: Creating a Generator Matrix (1/2)

```
1  int main(int argc, const char* argv[]) {
2
3          CodeMatrix oGenerator;
4
5          oGenerator.Build(&BuildFunction,512);
6
7          oGenerator.PrintMatrix("sha1me.cm");
8          exit(1);
9  }
```

Example: Creating a Generator Matrix (2/2)

```
1  CodeWord BuildFunction(uint64_t & i) {
2
3          CodeWord oCodeWord;
4          uint32_t m[60];
5          uint32_t unitv = 1;
6
7          // 512 bit message block
8          for(uint32_t j = 0; j<16; j++)
9                  m[j] = 0;
10
11         // create i-th unit vector for the input
12         unitv = ROTR(unitv , i+1);
13
14         // set input to i-th unit vector
15         m[i/32] = unitv;
16
17         // call the message expansion
18         SHA1ME(m);
19
20         // add message to the code
21         for(uint32_t j = 0; j<60; j++)
22                 oCodeWord.Push32(m[j]);
23
24         return oCodeWord;
25 }
```

Example: Searching for Low Hamming Weight

```
1  int main(int argc, const char* argv[]) {
2
3          CodeMatrix oGenerator;
4          CodeWord oCodeWord;
5          Parameters oParameters;
6          InputHandler oInputHandler(oParameters);
7          LowWeightSearch oLowWS;
8
9          string sCMFile = "";
10
11         // parse the command line arguments
12         if(oInputHandler.ParseSettings(argc, argv))
13                 exit(-1);
14         // get the file name of the code matrix
15         sCMFile = oParameters.GetStringParameter(Parameters::CMFILE);
16         // read data from the file
17         oGenerator.ReadFromFile(sCMFile);
18         // start the search
19         oCodeWord = oLowWS.CanteautChabaud(oGenerator,oParameters);
20
21         // print the code word and the Hamming weight
22         oCodeWord.Print64();
23         cout << "Hamming weight is " << oCodeWord.GetHammingWeight() << endl;
24         exit(1);
25 }
```

Example: All in one (1/2)

```
1  int main(int argc, const char* argv[]) {
2
3          CodeMatrix oGenerator;
4          CodeWord oCodeWord;
5          Parameters oParameters;
6          InputHandler oInputHandler(oParameters);
7          LowWeightSearch oLowWS;
8
9          // add a custom parameter
10         bool bShortening = false;
11         oParameters.AddParameter("-f",0,"enable code shortening");
12
13         // parse the command line arguments
14         // example: ./allinone -i 100 -o sha1me.cw -f 1
15         if(oInputHandler.ParseSettings(argc, argv))
16                 exit(-1);
17
18         bShortening = oParameters.GetIntegerParameter("-f");
```

Example: All in one (2/2)

```
1   // use the build function to create the generator matrix
2   // for the last 60 words of the SHA1 m.e.
3   oGenerator.Build(&BuildFunction,512);
4
5   // if shortening is enabled...
6   if(bShortening) {
7       vector<uint64_t> vForceZero;
8       for(uint32_t i = 0; i < 32; i++)
9               vForceZero.push_back(oGenerator.GetColumns()-32+i);
10      oGenerator = LowWeightSearch::CodeShortening(oGenerator,vForceZero);
11  }
12
13  oCodeWord = oLowWS.CanteautChabaud(oGenerator,oParameters);
14  oCodeWord.Print64();
15  cout << "Hamming weight is " << oCodeWord.GetHammingWeight() << endl;
16
17  exit(1);
18 }
```

Conclusions

- First open source implementation of the CC-algorithm
- Toolbox with many useful functionalities
- Good performance (will be improved in future versions)
- Extensibility (different search algorithms, adding features)
- Easy usage and fast results
- Available to everybody
  `http://www.iaik.tugraz.at/content/research/krypto/codingtool/`

Thank you for your attention!

http://www.iaik.tugraz.at/content/research/krypto/codingtool/

References I

[1] Eric Brier, Shahram Khazaei, Willi Meier, and Thomas Peyrin.
Linearization Framework for Collision Attacks: Application to CubeHash and MD6.
Cryptology ePrint Archive, Report 2009/382, 2009.
urlhttp://eprint.iacr.org/.

[2] Anne Canteaut and Florent Chabaud.
A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511.
IEEE Transactions on Information Theory, 44(1):367–378, 1998.

[3] Florent Chabaud and Antoine Joux.
Differential Collisions in SHA-0.
In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[4] Sebastiaan Indesteege and Bart Preneel.
Practical Collisions for EnRUPT.
In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2009.

References II

[5] Florian Mendel and Tomislav Nad.
A Distinguisher for the Compression Function of SIMD-512.
In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 219–232. Springer, 2009.

[6] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen.
Exploiting Coding Theory for Collision Attacks on SHA-1.
In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 78–95. Springer, 2005.

[7] Vincent Rijmen and Elisabeth Oswald.
Update on SHA-1.
In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2005.