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

1. Introduction
2. The CodingTool Library
3. Examples
4. Conclusions
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)
  
  $G = (g_0, \ldots, g_k)^T$

  $c = x \cdot G$ 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

- 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)

```c
int main(int argc, const char* argv[]) {
    CodeMatrix oGenerator;
    oGenerator.Build(&BuildFunction,512);
    oGenerator.PrintMatrix("sha1me.cm");
    exit(1);
}
```
Example: Creating a Generator Matrix (2/2)

```c
CodeWord BuildFunction(uint64_t & i) {
    CodeWord oCodeWord;
    uint32_t m[60];
    uint32_t unitv = 1;

    // 512 bit message block
    for(uint32_t j = 0; j<16; j++)
        m[j] = 0;

    // create i-th unit vector for the input
    unitv = ROTR(unitv, i+1);

    // set input to i-th unit vector
    m[i/32] = unitv;

    // call the message expansion
    SHA1ME(m);

    // add message to the code
    for(uint32_t j = 0; j<60; j++)
        oCodeWord.Push32(m[j]);

    return oCodeWord;
}
```
Example: Searching for Low Hamming Weight

```c++
int main(int argc, const char* argv[]) {

    CodeMatrix oGenerator;
    CodeWord oCodeWord;
    Parameters oParameters;
    InputHandler oInputHandler(oParameters);
    LowWeightSearch oLowWS;

    string sCMFile = "";

    // parse the command line arguments
    if (oInputHandler.ParseSettings(argc, argv))
        exit(-1);

    // get the file name of the code matrix
    sCMFile = oParameters.GetStringParameter(Parameters::CMFILE);

    // read data from the file
    oGenerator.ReadFile(sCMFile);

    // start the search
    oCodeWord = oLowWS.CanteautChabaud(oGenerator, oParameters);

    // print the code word and the Hamming weight
    oCodeWord.Print64();
    cout << "Hamming weight is " << oCodeWord.GetHammingWeight() << endl;
    exit(1);
}
```
Example: All in one (1/2)

```c
int main(int argc, const char* argv[]) {
    CodeMatrix oGenerator;
    CodeWord oCodeWord;
    Parameters oParameters;
    InputHandler oInputHandler(oParameters);
    LowWeightSearch oLowWS;

    // add a custom parameter
    bool bShortening = false;
    oParameters.AddParameter("-f", 0, "enable code shortening");

    // parse the command line arguments
    // example: ./allinone -i 100 -o sha1me.cw -f 1
    if (oInputHandler.ParseSettings(argc, argv))
        exit(-1);

    bShortening = oParameters.GetIntegerParameter("-f");
}
```
Example: All in one (2/2)

```cpp
// use the build function to create the generator matrix
// for the last 60 words of the SHA1 m.e.
oGenerator.Build(&BuildFunction, 512);

// if shortening is enabled...
if (bShortening) {
    vector<uint64_t> vForceZero;
    for (uint32_t i = 0; i < 32; i++)
        vForceZero.push_back(oGenerator.GetColumns() - 32 + i);
    oGenerator = LowWeightSearch::CodeShortening(oGenerator, vForceZero);
}

oCodeWord = oLowWS.CanteautChabaud(oGenerator, oParameters);
oCodeWord.Print64();
cout << "Hamming weight is " << oCodeWord.GetHammingWeight() << endl;
exit(1);
```
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/](http://www.iaik.tugraz.at/content/research/krypto/codingtool/)
Thank you for your attention!

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


References II

A Distinguisher for the Compression Function of SIMD-512.  

Exploiting Coding Theory for Collision Attacks on SHA-1.  

Update on SHA-1.  