



Institut
Mines-Télécom

Some techniques of code obfuscation

SR2I301

Jean-Luc Danger

Télécom ParisTech, LTCI



Objectives and Principles

□ Motivations

- Prevent reverse engineering
- Protect secret data or operation

□ Target of transformations

- Data :
 - Constant Data
 - Variable Data
- Code

Banescu, S., & Pretschner, A. (2018). A tutorial on software obfuscation. *Advances in Computers*. Elsevier.

Constant Data: Encode literals

Listing 1: Code before Encode Literals

```
1 int main(int ac, char* av[]) {
2     int a = 1;
3     // do stuff
4     return 0;
5 }
```

Listing 2: Code after Encode Literals

```
1 int main(int ac, char* av[]) {
2     double s = sin(atof(av[1]));
3     double c = cos(atof(av[1]));
4     int a = (int) (s * s + c * c);
5     // do stuff
6     return 0;
7 }
```

The adversary can think 'a' is variable as $av[]$ is used in $\cos^2 + \sin^2$

Constant Data: White Box Cryptography

The constant is coded in Look Up Table (LUT)

Listing 4: 8-bit XOR with MSB of secret key

```
1 char xor(char input) {  
2   return input ^ 0x53;  
3 }
```

Listing 5: LUT-based 8-bit XOR

```
1 char lut[256] = {  
2   0x53, 0x52, 0x51, ..., 0x5C,  
3   0x43, 0x42, 0x41, ..., 0x4C,  
4   0x73, 0x72, 0x71, ..., 0x7C,  
5   |                                     |  
6   0xA3, 0xA2, 0xA1, ..., 0xAC  
7 };  
8  
9 char xor(char input) {  
10  return lut[input];  
11 }
```

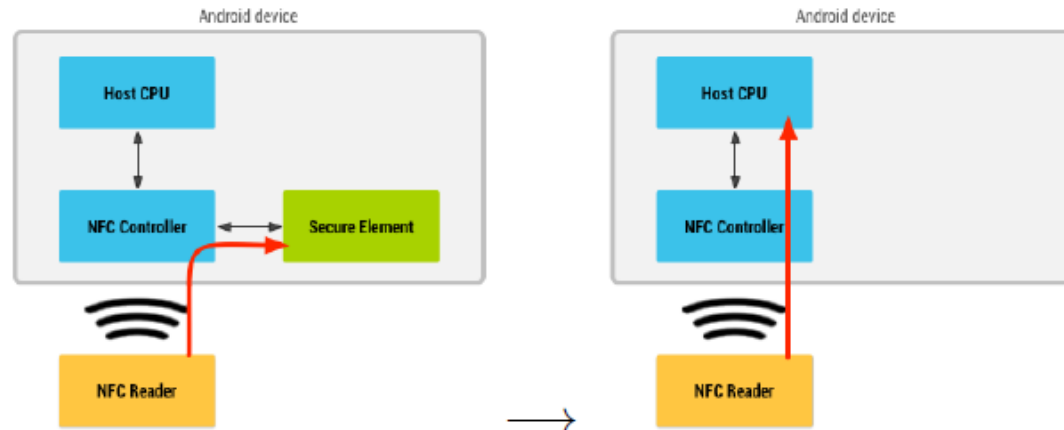
$f(x) \Rightarrow \text{LUT}(f(x))$

$f \circ g \circ h(x) \Rightarrow \text{LUT}(f \circ g \circ h(x))$ with **K** hidden

↑
Constant **K** (as a cryptographic key)

Motivation for White Box in Crypto

▶ Host Card Emulation (HCE):



▶ DRM, software protection, mobile payment

Use case: AES as an asymmetric key crypto primitive

	Public key	Private key
RSA	(e, N)	(d, p, q)
WBC	Obfuscated program, which implements AES_k : <ul style="list-style-type: none"> ▶ in: 128-bit plaintext ▶ out: 128-bit ciphertext 	k , 128-bit AES key

Variable Data: split or merge

SPLIT : 1 variable => N variables

MERGE : N variables => 1 variable

Listing 6: Split variable

```
1 char b1, b2, b3, b4; // i = b1,b2,b3,b4
2
3 int add(int a) {
4     return a + ((b1 << 8 + b2)
5                 << 8 + b3)
6                 << 8 + b4;
7 }
```

Listing 7: Merged variables

```
1 int i; // i = b1,b2,b3,b4
2 char add_b1(char a) {
3     return a + (i >> 24);
4 }
5 //...
6 char add_b4(char a) {
7     return a + (i & 0xff);
8 }
```

Code : Instruction substitution

Listing 20: Code before Encode Arithmetic

```
1 int main(int ac, char* av[]) {
2     int x = atoi(av[1]);
3     int y = atoi(av[2]);
4     int w = atoi(av[3]);
5     int z = x + y + w;
6     // do stuff
7     return 0;
8 }
```

Listing 21: Code after Encode Arithmetic

```
1 int main(int ac, char* av[]) {
2     int x = atoi(av[1]);
3     int y = atoi(av[2]);
4     int w = atoi(av[3]);
5     int z = (((x ^ y) + ((x & y) << 1)) | w) +
6             (((x ^ y) + ((x & y) << 1)) & w);
7     // do stuff
8     return 0;
9 }
```

Code: Garbage code

Listing 22: Code before inserting garbage code

```
1 int sum = 0;
2 for (i = 0; i < arr_len; i++)
3     sum += arr[i];
4 int average = sum / arr_len;
```

Listing 23: Code after inserting garbage code

```
1 int sum = 0;
2 int prod = 1;
3 for (i = 0; i < arr_len; i++) {
4     sum += arr[i];
5     prod *= arr[i];
6 }
7 int average = sqrt(prod);
8 average = sum / arr_len;
```

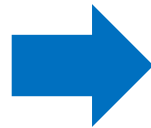

Code: Dead code

```
1  /* Original code */  
2  Stmt1; ... Stmtn;
```

```
1  /* With dead code */  
2  Stmt1; Stmt2; Stmt3; Stmt4;  
3  if (opaque_is_always_true)  
4      { /* Alive code */  
5          Stmt5; ... Stmtn;  
6      }  
7  else  
8      { /* Dead code */  
9          Stmtn+1; ... Stmtm;  
10     }
```

Code: flattening

```
int fct(int a, int b)
  int r=-1;
  if (a<b) {
    r=a;
  }
  else {
    if (a==b) {
      r=0;
    }
    else {
      r=b;
    }
  }
  return r;
}
```



```
int flt(int a, int b) {
  int r,x=0;
  while (1) switch (x) {
    case 0: r=-1; x = (a<b)?1:2; break;
    case 1: r=a; x=5; break;
    case 2: x = (a==b)?3:4; break;
    case 3: r=0; x=5; break;
    case 4: r=b; x=5; break;
    case 5: return r;
  }
}
```

Code : Tiles Loops

```
1  /* Original loop */
2  for(i=0; i<N; ++i){
3      ...
4  }
```

```
1  /* Tiled loop */
2  /* split in blocks of size B */
3  for(j=0; j<N; j+=B)
4      for(i=j; i<min(N, j+B); ++i){
5          ...
6      }
```

Conclusions

❑ Many more methods, Refer to:

Banescu, S., & Pretschner, A. (2018). A tutorial on software obfuscation. *Advances in Computers*. Elsevier.

- ❑ But they are all prone to attack
- ❑ They just slow down the attack process
- ❑ With a significant increase in complexity and time