Symmetric-key cryptography VI

Example of cryptanalysis of block ciphers

Tanja Lange
(with lots of slides by Daniel J. Bernstein)

Eindhoven University of Technology

2MMC10 – Cryptology

TEA

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

XORTEA: a bad cipher, not proposed

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x ^= y^c ^ (y<<4)^k[0]
             ^ (y>>5)^k[1];
    y ^= x^c ^ (x<<4)^k[2]
             ^ (x>>5)^k[3];
  }
  b[0] = x; b[1] = y;
}
```

XORTEA: a bad cipher, not proposed

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x ^= y^c ^ (y<<4)^k[0]
             ^ (y>>5)^k[1];
    y ^= x^c ^ (x<<4)^k[2]
             ^ (x>>5)^k[3];
  }
  b[0] = x; b[1] = y;
}
```

Output bits are linear functions of input bits!

XORTEA: a bad cipher, not proposed

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x ^= y^c ^ (y<<4)^k[0]
             ^ (y>>5)^k[1];
    y ^= x^c ^ (x<<4)^k[2]
             ^ (x>>5)^k[3];
  }
  b[0] = x; b[1] = y;
}
```

Output bits are linear functions of input bits!

e.g. bit b_0 in output is

$1 \oplus k_0 \oplus k_1 \oplus k_3 \oplus k_{10} \oplus k_{11} \oplus k_{12} \oplus k_{20} \oplus k_{21} \oplus k_{30} \oplus k_{32} \oplus k_{33} \oplus k_{35} \oplus k_{42} \oplus k_{43} \oplus k_{44} \oplus k_{52} \oplus k_{53} \oplus k_{62} \oplus k_{64} \oplus k_{67} \oplus k_{69} \oplus k_{76} \oplus k_{85} \oplus k_{94} \oplus k_{96} \oplus k_{99} \oplus k_{101} \oplus k_{108} \oplus k_{117} \oplus k_{126} \oplus b_1 \oplus b_3 \oplus b_{10} \oplus b_{12} \oplus b_{21} \oplus b_{30} \oplus b_{32} \oplus b_{33} \oplus b_{35} \oplus b_{37} \oplus b_{39} \oplus b_{42} \oplus b_{43} \oplus b_{44} \oplus b_{47} \oplus b_{52} \oplus b_{53} \oplus b_{57} \oplus b_{62}.$

XORTEA: a bad cipher, not proposed

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x ^= y^c ^ (y<<4)^k[0]
             ^ (y>>5)^k[1];
    y ^= x^c ^ (x<<4)^k[2]
             ^ (x>>5)^k[3];
  }
  b[0] = x; b[1] = y;
}
```

Output bits are linear functions of input bits!

e.g. bit b_0 in output is

$1 \oplus k_0 \oplus k_1 \oplus k_3 \oplus k_{10} \oplus k_{11} \oplus k_{12} \oplus k_{20} \oplus k_{21} \oplus k_{30} \oplus k_{32} \oplus k_{33} \oplus k_{35} \oplus k_{42} \oplus k_{43} \oplus k_{44} \oplus k_{52} \oplus k_{53} \oplus k_{62} \oplus k_{64} \oplus k_{67} \oplus k_{69} \oplus k_{76} \oplus k_{85} \oplus k_{94} \oplus k_{96} \oplus k_{99} \oplus k_{101} \oplus k_{108} \oplus k_{117} \oplus k_{126} \oplus b_1 \oplus b_3 \oplus b_{10} \oplus b_{12} \oplus b_{21} \oplus b_{30} \oplus b_{32} \oplus b_{33} \oplus b_{35} \oplus b_{37} \oplus b_{39} \oplus b_{42} \oplus b_{43} \oplus b_{44} \oplus b_{47} \oplus b_{52} \oplus b_{53} \oplus b_{57} \oplus b_{62}.$

Easy to distinguish from random permutation, not PRP.

XORTEA: a bad cipher, not proposed

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x ^= y^c ^ (y<<4)^k[0]
             ^ (y>>5)^k[1];
    y ^= x^c ^ (x<<4)^k[2]
             ^ (x>>5)^k[3];
  }
  b[0] = x; b[1] = y;
}
```

Output bits are linear functions of input bits!

e.g. bit b_0 in output is

$1 \oplus k_0 \oplus k_1 \oplus k_3 \oplus k_{10} \oplus k_{11} \oplus k_{12} \oplus k_{20} \oplus k_{21} \oplus k_{30} \oplus k_{32} \oplus k_{33} \oplus k_{35} \oplus k_{42} \oplus k_{43} \oplus k_{44} \oplus k_{52} \oplus k_{53} \oplus k_{62} \oplus k_{64} \oplus k_{67} \oplus k_{69} \oplus k_{76} \oplus k_{85} \oplus k_{94} \oplus k_{96} \oplus k_{99} \oplus k_{101} \oplus k_{108} \oplus k_{117} \oplus k_{126} \oplus b_1 \oplus b_3 \oplus b_{10} \oplus b_{12} \oplus b_{21} \oplus b_{30} \oplus b_{32} \oplus b_{33} \oplus b_{35} \oplus b_{37} \oplus b_{39} \oplus b_{42} \oplus b_{43} \oplus b_{44} \oplus b_{47} \oplus b_{52} \oplus b_{53} \oplus b_{57} \oplus b_{62}.$

Easy to distinguish from random permutation, not PRP.

For blocks B_1, B_2, B_3, B_4 with $B_1 \oplus B_2 = B_3 \oplus B_4$ have

$$\underset{k}{\mathrm{XORTEA}}(B_1) \oplus \underset{k}{\mathrm{XORTEA}}(B_2) = \underset{k}{\mathrm{XORTEA}}(B_3) \oplus \underset{k}{\mathrm{XORTEA}}(B_4).$$

TEA

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

LEFTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y<<5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x<<5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

LEFTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y<<5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x<<5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Addition is not $\mathbf{F}_2$-linear,
but addition mod 2 is $\mathbf{F}_2$-linear.
Bit b_0 in output (after 32 rounds):

$1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.

Easy to distinguish, not PRP.

LEFTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y<<5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x<<5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Addition is not $\mathbf{F}_2$-linear,
but addition mod 2 is $\mathbf{F}_2$-linear.
Bit b_0 in output (after 32 rounds):

$1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.
Easy to distinguish, not PRP.

Higher output bits, $b_i, i > 0$,
are increasingly nonlinear
but they never affect first bit.

LEFTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y<<5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x<<5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Addition is not $\mathbf{F}_2$-linear,
but addition mod 2 is $\mathbf{F}_2$-linear.
Bit b_0 in output (after 32 rounds):

$1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.
Easy to distinguish, not PRP.

Higher output bits, $b_i, i > 0$,
are increasingly nonlinear
but they never affect first bit.

In TEA, >>5 **diffuses** nonlinear
changes from high bits to low bits.

LEFTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y<<5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x<<5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Addition is not $\mathbf{F}_2$-linear,
but addition mod 2 is $\mathbf{F}_2$-linear.
Bit b_0 in output (after 32 rounds):

$1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.
Easy to distinguish, not PRP.

Higher output bits, $b_i, i > 0$,
are increasingly nonlinear
but they never affect first bit.

In TEA, >>5 **diffuses** nonlinear changes from high bits to low bits.

(Diffusion from low bits to high bits: <<4; carries in addition.)

TEA

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

TEA4: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 4;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

TEA4: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 4;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Fast attack:
$\mathrm{TEA4}_k(x+2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same bit b_0 in output.

TEA4: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 4;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Fast attack:
$\mathrm{TEA4}_k(x+2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same bit b_0 in output.

Trace x, y differences through steps in computation.
$r = 0$: multiples of $2^{31}, 2^{26}$.
$r = 1$: multiples of $2^{21}, 2^{16}$.
$r = 2$: multiples of $2^{11}, 2^{6}$.
$r = 3$: multiples of $2^{1}, 2^{0}$.

TEA4: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 4;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Fast attack:
$\mathrm{TEA4}_k(x+2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same bit b_0 in output.

Trace x, y differences through steps in computation.
$r = 0$: multiples of $2^{31}, 2^{26}$.
$r = 1$: multiples of $2^{21}, 2^{16}$.
$r = 2$: multiples of $2^{11}, 2^{6}$.
$r = 3$: multiples of $2^{1}, 2^{0}$.

Uniform random function F:
$F(x + 2^{31}, y)$ and $F(x, y)$ have same first bit with probability $1/2$.

TEA4: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 4;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Fast attack:
$\mathrm{TEA4}_k(x+2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same bit b_0 in output.

Trace x, y differences through steps in computation.
$r = 0$: multiples of $2^{31}, 2^{26}$.
$r = 1$: multiples of $2^{21}, 2^{16}$.
$r = 2$: multiples of $2^{11}, 2^{6}$.
$r = 3$: multiples of $2^{1}, 2^{0}$.

Uniform random function F:
$F(x + 2^{31}, y)$ and $F(x, y)$ have same first bit with probability $1/2$.

PRF advantage $1/2$. Two pairs (x, y): advantage $3/4$. ...

TEA4: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 4;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Fast attack:
$\mathrm{TEA4}_k(x+2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same bit b_0 in output.

Trace x, y differences through steps in computation.
$r = 0$: multiples of $2^{31}, 2^{26}$.
$r = 1$: multiples of $2^{21}, 2^{16}$.
$r = 2$: multiples of $2^{11}, 2^{6}$.
$r = 3$: multiples of $2^{1}, 2^{0}$.

Uniform random function F:
$F(x + 2^{31}, y)$ and $F(x, y)$ have same first bit with probability $1/2$.

PRF advantage $1/2$. Two pairs (x, y): advantage $3/4$. ...

This is a small example of a differential attack.

TEA

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

REPTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0x9e3779b9;
  for (r = 0;r < 1000;r += 1) {
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

REPTEA: another bad cipher

$\mathrm{REPTEA}_k(b) = l_k^{1000}(b)$
where l_k does `x+=...;y+=...`.

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0x9e3779b9;
  for (r = 0;r < 1000;r += 1) {
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

REPTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0x9e3779b9;
  for (r = 0;r < 1000;r += 1) {
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

$\mathrm{REPTEA}_k(b) = l_k^{1000}(b)$
where l_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.

REPTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0x9e3779b9;
  for (r = 0;r < 1000;r += 1) {
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

$\mathrm{REPTEA}_k(b) = l_k^{1000}(b)$
where l_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.

Good chance that some b in list also has $a = l_k(b)$ in list. Then $\mathrm{REPTEA}_k(a) = l_k(\mathrm{REPTEA}_k(b))$.

REPTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0x9e3779b9;
  for (r = 0;r < 1000;r += 1) {
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

$\mathrm{REPTEA}_k(b) = l_k^{1000}(b)$
where l_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.

Good chance that some b in list also has $a = l_k(b)$ in list. Then $\mathrm{REPTEA}_k(a) = l_k(\mathrm{REPTEA}_k(b))$.

For each (b, a) from list:
Try solving equations $a = l_k(b)$, $\mathrm{REPTEA}_k(a) = l_k(\mathrm{REPTEA}_k(b))$ to figure out k. Test k.
Move to next pair if k fails.

REPTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k)
{
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0x9e3779b9;
  for (r = 0;r < 1000;r += 1) {
    x += y+c ^ (y<<4)+k[0]
             ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2]
             ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

$\mathrm{REPTEA}_k(b) = I_k^{1000}(b)$
where I_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.

Good chance that some b in list also has $a = I_k(b)$ in list. Then $\mathrm{REPTEA}_k(a) = I_k(\mathrm{REPTEA}_k(b))$.

For each (b, a) from list:
Try solving equations $a = I_k(b)$, $\mathrm{REPTEA}_k(a) = I_k(\mathrm{REPTEA}_k(b))$ to figure out k. Test k.
Move to next pair if k fails.

This is a **slide attack.** TEA avoids this by varying c.