

## Aufgabe 25

(a)  $G$  ist gegeben durch

$$\begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

( $a_i$  sind Koeffizienten des Eingabepolynoms,  $b_i$  Koeffizienten des Ausgabepolynoms) Das Eingabepolynom zu  $G$  ist das multiplikative Inverse des Eingabepolynoms zu  $S$ .

Für  $\pi_S(02)$  gilt demnach  $a = \text{Inv}(00000010)^T = (10001101)^T$  ( $\text{Inv}$  mittels *extended\_euclid* berechnet) und somit

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Also  $b = (01110111)^T$ , was 77 Hex entspricht. Für  $\pi_S(06)$  ( $a = \text{Inv}(00000110)^T = (01111011)^T$ ) erfolgt die Rechnung analog :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\
\Rightarrow \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Also  $b = (01101111)^T$ , was 6F Hex entspricht.

- (b) Auf dem letzten Übungszettel hatten wir MixColumns schon mittels XOR und einer Funktion  $xtime(a)$  implementiert; diese lässt sich natürlich auch als table-lookup realisieren, wie es auch im Standard unter 'Implementation aspects' vorgeschlagen ist (zwar ist ein shift-left und anschließendes conditional XOR programmatisch schoener, sicherheitstechnisch jedoch problematischer, da hier ggf. verschieden viele Cycles benutzt werden, was gewisse Side-Channel Attacken vereinfacht).

Warum die im Rijndael-Paper (z.B. unter [HTTP://CSRC.NIST.GOV/ENCRYPTION/AES/RIJNDAEL/RIJNDAEL.PDF](http://csrc.nist.gov/encryption/aes/rijndael/rijndael.pdf)) angegebene Methode funktioniert, soll hier Anhand einer einzelnen Spalte hergeleitet werden; die anderen vier verhalten sich analog hierzu.

MixColumns für eine Spalte ist definiert durch

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

hieraus ergibt sich

$$\begin{aligned} b_0 &= 02 \cdot a_0 + 03 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3 \\ b_1 &= 01 \cdot a_0 + 02 \cdot a_1 + 03 \cdot a_2 + 01 \cdot a_3 \\ b_2 &= 01 \cdot a_0 + 01 \cdot a_1 + 02 \cdot a_2 + 03 \cdot a_3 \\ b_3 &= 03 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + 02 \cdot a_3 \end{aligned}$$

Durch einfache Umformungen erhält man

$$\begin{aligned}
b_0 &= 02 \cdot (a_0 + a_1) + 01 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3 \\
b_1 &= 01 \cdot a_0 + 02 \cdot (a_1 + ta_2) + 01 \cdot a_2 + 01 \cdot a_3 \\
b_2 &= 01 \cdot a_0 + 01 \cdot a_1 + 02 \cdot (a_2 + a_3) + 01 \cdot a_3 \\
b_3 &= 01 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + 02 \cdot (a_3 + a_0)
\end{aligned}$$

Mittels einer Operation  $xtime(a)$ , welche ein Polynom in  $GF(2^8)$  mit 02 multipliziert (sowie der Tatsache, dass  $01 \cdot a = a$ ) lässt sich dies auch als

$$\begin{aligned}
b_0 &= xtime(a_0 + a_1) + 01 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3 \\
b_1 &= 01 \cdot a_0 + xtime(a_1 + ta_2) + 01 \cdot a_2 + 01 \cdot a_3 \\
b_2 &= 01 \cdot a_0 + 01 \cdot a_1 + xtime(a_2 + a_3) + 01 \cdot a_3 \\
b_3 &= 01 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + xtime(a_3 + a_0)
\end{aligned}$$

darstellen. Da  $+$  in  $GF(2^8)$  einem XOR auf Bitebene entspricht, ist hiermit eine Methode gegeben, MixColumns mittels XOR und einem Table-Lookup (nämlich  $xtime(a)$ ) zu implementieren; es werden in dieser trivialen Version je Spalte 16 XORs (also insgesamt 64 XORs je Runde) benötigt; Dies kann jedoch noch weiter reduziert werden :

$$\begin{aligned}
b_0 &= 01 \cdot a_0 + xtime(a_0 + a_1) + 01 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3 \\
b_1 &= 01 \cdot a_1 + xtime(a_1 + a_2) + 01 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3 \\
b_2 &= 01 \cdot a_2 + xtime(a_2 + a_3) + 01 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3 \\
b_3 &= 01 \cdot a_3 + xtime(a_3 + a_0) + 01 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3
\end{aligned}$$

Mittels einer temporären Variable ergibt sich daraus

$$\begin{aligned}
Tmp &= 01 \cdot a_0 + 01 \cdot a_1 + 01 \cdot a_2 + 01 \cdot a_3 \\
b_0 &= 01 \cdot a_0 + xtime(a_0 + a_1) + Tmp \\
b_1 &= 01 \cdot a_1 + xtime(a_1 + a_2) + Tmp \\
b_2 &= 01 \cdot a_2 + xtime(a_2 + a_3) + Tmp \\
b_3 &= 01 \cdot a_3 + xtime(a_3 + a_0) + Tmp
\end{aligned}$$

, was einer Reduktion auf 15 XORs je Spalte (60 je Runde) gleichkommt (und der im Rijndael-Paper vorgeschlagenen Methode für 8-Bit Computer entspricht).  $xtime(a)$  an sich lässt sich entweder als Table-Lookup (mit 256 Einträgen, da die Multiplikation ja eine bijektive Abbildung in  $GF(2^8)$  ist) oder aber als shift-left und einem anschließenden XOR mit 1B implementieren (Rationale siehe Rijndael paper). Ein table-lookup ist schneller, benötigt aber natürlich auch mehr Speicher.

Wir haben unsere Implementation von MixColumns so angepasst; sie ist im Anhang zu finden.

## Aufgabe 26

(a) Die kleinste natürliche Zahl, die Reste 1 (,2,3,5) lässt, wenn man sie durch 3 (5,7,11) teilt ist 577. (Perlroutine CHINESEREST) (der Teiler 9 wurde hier weggelassen, da er nicht paarweise teilefremd zu den anderen gegebenen Teilern ist (3|9))

(b)

$$\begin{aligned} 3^{21} \bmod 7 &= 6 \\ 5^{18} \bmod 11 &= 4 \end{aligned}$$

(Perlroutine MODEXP)

(c) 1111 ist sicher nicht prim, ein Zeuge ist 2 (Perlroutine PSEUDOPRIME)

(d)  $x^2 \equiv 17 \pmod{19}$  hat zwei Lösungen : -6 und 6 (Perlroutine P26D, parametrisiert)

(e) Verfahren:

(a) Löse  $2x_k t \equiv \frac{a-x_k^2}{p^k} \pmod{p^{k+1}}$  nach  $t$

(b) Setze  $t$  in  $x_{k+1} = (x_k + p^k \cdot t)$  ein

(c) Für  $x_{k+1}$  gilt dann  $x_{k+1}^2 \equiv a \pmod{p^{k+1}}$

Berechnet man  $x_1 \equiv a \pmod{p}$  auf herkömmliche Weise, kann man durch mehrfache Iteration der oben angegebenen Schritte die Lösung für beliebige Potenzen  $k$  von  $p$  erhalten.

Beweis für die Korrektheit des Verfahrens:

(Es gelte  $x_k \equiv a \pmod{p^k}$  und  $-n$  bezeichne das additive Inverse zu  $n$ .)

Wir betrachten die Kongruenz  $(x_k + p^k \cdot t)^2 \stackrel{(1)}{\equiv} x_k^2 + 2x_k t p^k + p^{2k} \cdot t^2 \pmod{p^{k+1}}$ .

Wir wählen  $t$  (wie oben) so, dass gilt  $2x_k t \equiv \frac{a-x_k^2}{p^k} \pmod{p^{k+1}}$ . Eingesetzt ergibt sich dann  $(x_k + p^k \cdot t)^2 \equiv x_k^2 + p^k \cdot \frac{a-x_k^2}{p^k} \pmod{p^{k+1}}$  also  $(x_k + p^k \cdot t)^2 \equiv a \pmod{p^{k+1}}$ .

(1): Aus  $p^{2k} \cdot t^2 = p^{k+1} \cdot p^{k-1} \cdot t$  folgt  $[x_k^2 + 2x_k t p^k]_{p^{k+1}} \equiv [x_k^2 + 2x_k t p^k + p^{2k} \cdot t^2]_{p^{k+1}}$ , daher kann  $p^{2k} \cdot t^2$  weggelassen werden.

(2): Da  $x_k^2 = m \cdot p^k + a$  ergibt  $\frac{a-x_k^2}{p^k}$  eine ganze Zahl.

Dieses Verfahren ist in Perl in P26E implementiert.  $y^2 \equiv 17 \pmod{19^2} \Rightarrow y = 576$  sowie  $z^2 \equiv 17 \pmod{19^3} \Rightarrow z = 1317865$

Einige Ergebnisse (modexp, Chinesischer Restsatz, Kleiner Fermat, 26d, 26e) wurden mittels eines Perlskripts ermittelt :

./uebung7.pl

## Aufgabe 27

(a)  $e_K(7) = 7^{17} \bmod 3599 = 1698$

3599 ist das Produkt von 59 und 61. Hiermit ist es dann trivial,  $d$  zu berechnen : Da

$$d = e^{-1} \bmod \Phi(n)$$

gilt, und wir nun  $\Phi(n)$  kennen ( $58 * 60 = 3480$ ), muss noch  $e^{-1}$  berechnet werden :  $loese\_mod\_eq(17, 1, 3480) = 1433$ ; damit ist  $d$  gegeben und eine einfache Dechiffrierung mittels  $rsa(1901, 1433, 3599)$  ergibt 3592

Wir haben die noetigen Methoden implementiert :

```
./uebung7.pl
```

- (b) Der Schlüssel, von Bob scheint aus folgendem Grund gefälscht: Faktorisiert man  $n = 143$ , so ergeben sich für  $p$  und  $q$ ,  $p = 11$  und  $q = 13$  (oder umgekehrt). Somit für  $\Phi(n) = 12 \cdot 10 = 120$ . In RSA wird der Exponent  $e$  nun so gewählt, dass  $2 \leq e < \Phi(n)$  gilt. Also  $e < 120$ . Der Exponent des Schlüssels ist aber mit  $e = 121 > 120$  angegeben.

## Anhang

```
#!/usr/local/bin/perl -w
use strict; $|=1; no strict 'refs'; use POSIX; use Math::BigInt;

sub zettel7 {
  sub extended_euclid {
    $_[1] or return $_[0], 1, 0;
    my ($d, $x, $y) = extended_euclid ($_[1], $_[0] % $_[1]);
    $d, $y, $x - int ($_[0] / $_[1]) * $y;
  }

  sub loese_mod_eg {
    my ($d, $x, $y) = extended_euclid ($_[0], $_[2]);
    $_[1] % $d ? undef :
    map { ($x * $_[1] / $d % $_[2] + $_ * $_[2] / $d) % $_[2] } 0..$d-1;
  }

  sub modexp {
    my ($a, $b, $n) = @_;
    my $d = 1;
    my @b = split //, sprintf "%0b", $b;

    for my $i (0..@b-1) {
      $d = $d*$d % $n;
      $d = $d*$a % $n if $b[$i]
    }
    $d
  }
}
```

```

}

sub pseudoprime {
    my $n = shift;
    for (1..$n-1) {
        return "Sicher nicht prim (Zeuge $_)" if modexp (_, $n-1, $n) != 1
    }
    "Wahrscheinlich prim"
}

sub chineseRest{
    my @rests = @{shift ()};
    my @modules = @{shift ()};

    die "On a lonely island" if (@rests - @modules);

    my $N = 1; $N *= $modules[$_] for 0..@modules-1;
    my @NI; $NI[$_] = $N/$modules [$_] for 0..@modules-1;

    my @c = @NI;
    for (0..@modules-1) {
        my @tmp = loese_mod_eg ($NI[$_], 1, $modules[$_]);
        $c[$_] *= $tmp [0];
    }

    my $result;
    $result += $rests[$_]*$c[$_] for 0..@modules-1;
    $result%$N
}

sub p26d {
    my ($a, $b) = @_;
    if (modexp ($a, ($b-1)/2, $b) + 1) {
        my $x = $a;
        while (floor(sqrt($x)) != sqrt($x)) { $x += $b }
        return sqrt($x), -sqrt($x)
    }
    undef
}

sub p26e {
    my ($a, $b, $d) = @_;
    my $c = [p26d ($a,$b)]->[0];
    for (1..$d-1) {
        $c = $c + $b**$_ * [loese_mod_eg (2 * $c, ($a - $c ** 2) /
            ($b ** $_), $b ** ($_+1))]->[0]
    }
    $c
}

```

```

sub rsa { $_[0]**$_[1]%$_[2] }

sub factorize {
    my $r;
    $r = $_[0]%$_ ? $r : $_ for 1..floor(sqrt($_[0]));
    $r, $_[0]/$r
}

print "Chineserest : ", chineseRest ([1,2,3,5], [3,5,7,11]), "\n";

print "3^21%7 : ", modexp (3, 21, 7), "\n";
print "5^18%11 : ", modexp (5, 18, 11), "\n";
print "Fermat : ", pseudoprime (1111), "\n";
print "x^2==17mod19 : ", join (" ", p26d (17,19)), "\n";
print "y^2==17mod19^2 : ", p26e (17,19,2), "\n";
print "z^2==17mod19^3 : ", p26e (17,19,3), "\n";
print "rsa (7,17,3599) : ", rsa (7,17,3599), "\n";
print "factorize (3599) : ", join (" ", factorize (3599)), "\n";
my $d = [loese_mod_eg (17, 1, ([[factorize (3599)]->[0]-1]*
    ([factorize (3599)]->[1]-1)))]->[0];
print "rsa (1901,$d,3599) : ", rsa (new Math::BigInt(1901),
    new Math::BigInt($d), new Math::BigInt(3599)), "\n";
}

sub mixColumns {
    my @old = @_;
    for my $i (0..3) {
        my $T = $old[0+$i] ^ $old[4+$i] ^ $old[8+$i] ^ $old[12+$i];
        $_[4*$i+$i] ^=
        {0x00 => 0x00, 0x01 => 0x02, 0x02 => 0x04, 0x03 => 0x06,
         0x04 => 0x08, 0x05 => 0x0a, 0x06 => 0x0c, 0x07 => 0x0e,
         0x08 => 0x10, 0x09 => 0x12, 0x0a => 0x14, 0x0b => 0x16,
         0x0c => 0x18, 0x0d => 0x1a, 0x0e => 0x1c, 0x0f => 0x1e,
         0x10 => 0x20, 0x11 => 0x22, 0x12 => 0x24, 0x13 => 0x26,
         0x14 => 0x28, 0x15 => 0x2a, 0x16 => 0x2c, 0x17 => 0x2e,
         0x18 => 0x30, 0x19 => 0x32, 0x1a => 0x34, 0x1b => 0x36,
         0x1c => 0x38, 0x1d => 0x3a, 0x1e => 0x3c, 0x1f => 0x3e,
         0x20 => 0x40, 0x21 => 0x42, 0x22 => 0x44, 0x23 => 0x46,
         0x24 => 0x48, 0x25 => 0x4a, 0x26 => 0x4c, 0x27 => 0x4e,
         0x28 => 0x50, 0x29 => 0x52, 0x2a => 0x54, 0x2b => 0x56,
         0x2c => 0x58, 0x2d => 0x5a, 0x2e => 0x5c, 0x2f => 0x5e,
         0x30 => 0x60, 0x31 => 0x62, 0x32 => 0x64, 0x33 => 0x66,
         0x34 => 0x68, 0x35 => 0x6a, 0x36 => 0x6c, 0x37 => 0x6e,
         0x38 => 0x70, 0x39 => 0x72, 0x3a => 0x74, 0x3b => 0x76,
         0x3c => 0x78, 0x3d => 0x7a, 0x3e => 0x7c, 0x3f => 0x7e,
         0x40 => 0x80, 0x41 => 0x82, 0x42 => 0x84, 0x43 => 0x86,
         0x44 => 0x88, 0x45 => 0x8a, 0x46 => 0x8c, 0x47 => 0x8e,
         0x48 => 0x90, 0x49 => 0x92, 0x4a => 0x94, 0x4b => 0x96,
         0x4c => 0x98, 0x4d => 0x9a, 0x4e => 0x9c, 0x4f => 0x9e,
         0x50 => 0xa0, 0x51 => 0xa2, 0x52 => 0xa4, 0x53 => 0xa6,
        }
    }
}

```

```

0x54 => 0xa8, 0x55 => 0xaa, 0x56 => 0xac, 0x57 => 0xae,
0x58 => 0xb0, 0x59 => 0xb2, 0x5a => 0xb4, 0x5b => 0xb6,
0x5c => 0xb8, 0x5d => 0xba, 0x5e => 0xbc, 0x5f => 0xbe,
0x60 => 0xc0, 0x61 => 0xc2, 0x62 => 0xc4, 0x63 => 0xc6,
0x64 => 0xc8, 0x65 => 0xca, 0x66 => 0xcc, 0x67 => 0xce,
0x68 => 0xd0, 0x69 => 0xd2, 0x6a => 0xd4, 0x6b => 0xd6,
0x6c => 0xd8, 0x6d => 0xda, 0x6e => 0xdc, 0x6f => 0xde,
0x70 => 0xe0, 0x71 => 0xe2, 0x72 => 0xe4, 0x73 => 0xe6,
0x74 => 0xe8, 0x75 => 0xea, 0x76 => 0xec, 0x77 => 0xee,
0x78 => 0xf0, 0x79 => 0xf2, 0x7a => 0xf4, 0x7b => 0xf6,
0x7c => 0xf8, 0x7d => 0xfa, 0x7e => 0xfc, 0x7f => 0xfe,
0x80 => 0x1b, 0x81 => 0x19, 0x82 => 0x1f, 0x83 => 0x1d,
0x84 => 0x13, 0x85 => 0x11, 0x86 => 0x17, 0x87 => 0x15,
0x88 => 0x0b, 0x89 => 0x09, 0x8a => 0x0f, 0x8b => 0x0d,
0x8c => 0x03, 0x8d => 0x01, 0x8e => 0x07, 0x8f => 0x05,
0x90 => 0x3b, 0x91 => 0x39, 0x92 => 0x3f, 0x93 => 0x3d,
0x94 => 0x33, 0x95 => 0x31, 0x96 => 0x37, 0x97 => 0x35,
0x98 => 0x2b, 0x99 => 0x29, 0x9a => 0x2f, 0x9b => 0x2d,
0x9c => 0x23, 0x9d => 0x21, 0x9e => 0x27, 0x9f => 0x25,
0xa0 => 0x5b, 0xa1 => 0x59, 0xa2 => 0x5f, 0xa3 => 0x5d,
0xa4 => 0x53, 0xa5 => 0x51, 0xa6 => 0x57, 0xa7 => 0x55,
0xa8 => 0x4b, 0xa9 => 0x49, 0xaa => 0x4f, 0xab => 0x4d,
0xac => 0x43, 0xad => 0x41, 0xae => 0x47, 0xaf => 0x45,
0xb0 => 0x7b, 0xb1 => 0x79, 0xb2 => 0x7f, 0xb3 => 0x7d,
0xb4 => 0x73, 0xb5 => 0x71, 0xb6 => 0x77, 0xb7 => 0x75,
0xb8 => 0x6b, 0xb9 => 0x69, 0xba => 0x6f, 0xbb => 0x6d,
0xbc => 0x63, 0xbd => 0x61, 0xbe => 0x67, 0xbf => 0x65,
0xc0 => 0x9b, 0xc1 => 0x99, 0xc2 => 0x9f, 0xc3 => 0x9d,
0xc4 => 0x93, 0xc5 => 0x91, 0xc6 => 0x97, 0xc7 => 0x95,
0xc8 => 0x8b, 0xc9 => 0x89, 0xca => 0x8f, 0xcb => 0x8d,
0xcc => 0x83, 0xcd => 0x81, 0xce => 0x87, 0xcf => 0x85,
0xd0 => 0xbb, 0xd1 => 0xb9, 0xd2 => 0xbf, 0xd3 => 0xbd,
0xd4 => 0xb3, 0xd5 => 0xb1, 0xd6 => 0xb7, 0xd7 => 0xb5,
0xd8 => 0xab, 0xd9 => 0xa9, 0xda => 0xaf, 0xdb => 0xad,
0xdc => 0xa3, 0xdd => 0xa1, 0xde => 0xa7, 0xdf => 0xa5,
0xe0 => 0xdb, 0xe1 => 0xd9, 0xe2 => 0xdf, 0xe3 => 0xdd,
0xe4 => 0xd3, 0xe5 => 0xd1, 0xe6 => 0xd7, 0xe7 => 0xd5,
0xe8 => 0xcb, 0xe9 => 0xc9, 0xea => 0xcf, 0xeb => 0xcd,
0xec => 0xc3, 0xed => 0xc1, 0xee => 0xc7, 0xef => 0xc5,
0xf0 => 0xfb, 0xf1 => 0xf9, 0xf2 => 0xff, 0xf3 => 0xfd,
0xf4 => 0xf3, 0xf5 => 0xf1, 0xf6 => 0xf7, 0xf7 => 0xf5,
0xf8 => 0xeb, 0xf9 => 0xe9, 0xfa => 0xef, 0xfb => 0xed,
0xfc => 0xe3, 0xfd => 0xe1, 0xfe => 0xe7, 0xff => 0xe5}
->{$old[4*$_+5i] ^ $old[(4*$_+4)%16+5i]} ^ $T for 0..3
}
@_
}

```

zettel7