Topics where Coding meets Crypto

January, 2015

SPCoding School, Campinas

Henk van Tilborg
Eindhoven University of Technology

1 Secret Sharing Schemes

1.1 The problem of how to share a secret

The designer of a new car model puts the drawings in a safe every evening. But he likes his colleagues to be able to continue his work, if something happens to him.

He does not trust any colleague enough to give him/her the combination. Instead, he wants to gives each of his 7 colleagues a "share" of the secret combination in such a way that:

1) Every 4-tuple of colleagues can recover the combination and open the safe.

2) Every 3-tuple of colleagues knows absolutely nothing about that combinaton.

Definition: An (n, k) threshold scheme is a secret sharing scheme among n participants in such a way that:

1) Every k-tuple of participants can recover the combination and open the safe.

2) Every (k-1)-tuple of participants knows absolutely nothing about that combination.

Applications:

Master keys that are used by banks to compute the personal secret keys on ATM cards. You do not want to store them at one location.

Codes that send missiles to other countries. You do not want to entrust a single person with them.

A possible complication: malicious colleagues.

1.2 An old, steel cabinet and a collection of padlocks

We start with an old, steel cabinet with an iron ring sticking out on which you can hang padlocks.

The colleagues are called *A*, *B*, *C*, *D*, *E*, *F* and *G*. Each has the keys of some of the padlocks.

Notice:

1) For every three colleagues there must be at least one padlock that none of them can open.

2) Each of the other colleagues must have the key of this padlock.

We put the names of the people who have the key of a particular padlock on that padlock in blue. In red we add the names of those who do not have the key of that padlock.

A, B, C, D have the key

E, F, G do not

Of course, you do not need two padlocks that both can be opened by *A*, *B*, *C*, *D* and not by *E*, *F*, *G*.

□ **How many padlocks do you need?**

The number of padlocks needed is the binomial coefficient $\binom{7}{4}$.

$$\binom{7}{4} = \frac{7\times6\times5\times4}{4\times3\times2\times1} = 35.$$

□ **How many keys is colleague *A*, and each of the others, carrying around?**

For every 3-tuple not including *A*, there is one padlock that they cannot open and of which *A* must have the key. So, the answer is $\binom{6}{3}$.

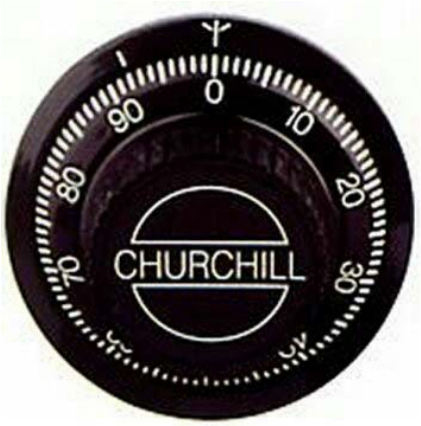
```
Binomial[6, 3]
```

```
20
```

This is all not very practical!

1.3 A safe with a number combination

We now look at safe with an old fashioned combination lock.



For example, the combination is given by:

19 - 82 - 63

Suppose that the owner/designer has two colleagues.

He could give

***A* the share** **1 _ 8 _ 6 _** **the left most digit of each numer, and**

***B* the share** **_ 9 _ 2 _ 3** **the right most digit of each number,**

This leaves only $10^3 = 1000$ possibilities for each of them to trt out, instead of the intended $10^6 = 1\,000\,000$. We do not want that.

1.3.1 A (2, 2) threshold scheme

A simple solution for two participants makes use of modulo 100 arithmetic: all calculations are done modulo 100.

***A* gets three random 2-digits long numbers:**

```
AA = RandomInteger[100, 3]
```

```
{27, 35, 91}
```

***B* also gets three 2-digits long number, but computed in such a way, that the two shares add up (coordinate-wise) to the combination 19, 82, 63 (mod 100).**

So

```
Secret = {19, 82, 63};
BB = Mod[Secret - AA, 100]
```

```
{92, 47, 72}
```

Indeed

```
Mod[AA + BB, 100]
```

```
{19, 82, 63}
```

Notice that you could as well have said that *B*'s share was generated in a random way and that *A*'s share was computed.

1.3.2 Also (3, 3), (4, 4) etc. threshold schemes

It is easy to generalize this approach to more participants in such a way that all are needed to recover the secret and that with one person missing the others do not know anything at all.

For example with three participants, you give *A* en *B* the random combinations:

```
AA = RandomInteger[100, 3]
BB = RandomInteger[100, 3]
```

```
{41, 12, 14}
```

```
{96, 5, 22}
```

The combination/share for *C* follows from

```
Secret = {19, 82, 63};
CC = Mod[Secret - AA - BB, 100]
```

```
{82, 65, 27}
```

Indeed

```
Mod[AA + BB + CC, 100]
```

```
{19, 82, 63}
```

This solution is clearly not flexible. You do not always want all participants to be present.

1.4 An (*n*, *k*) threshold scheme over ℝ

In this chapter we perform all calculations in the field ℝ of the real numbers. In the next chapter we correct that.

Let us focus on the first number in the safe combination, so on $S = 19$.

1.4.1 A (7, 2) threshold scheme

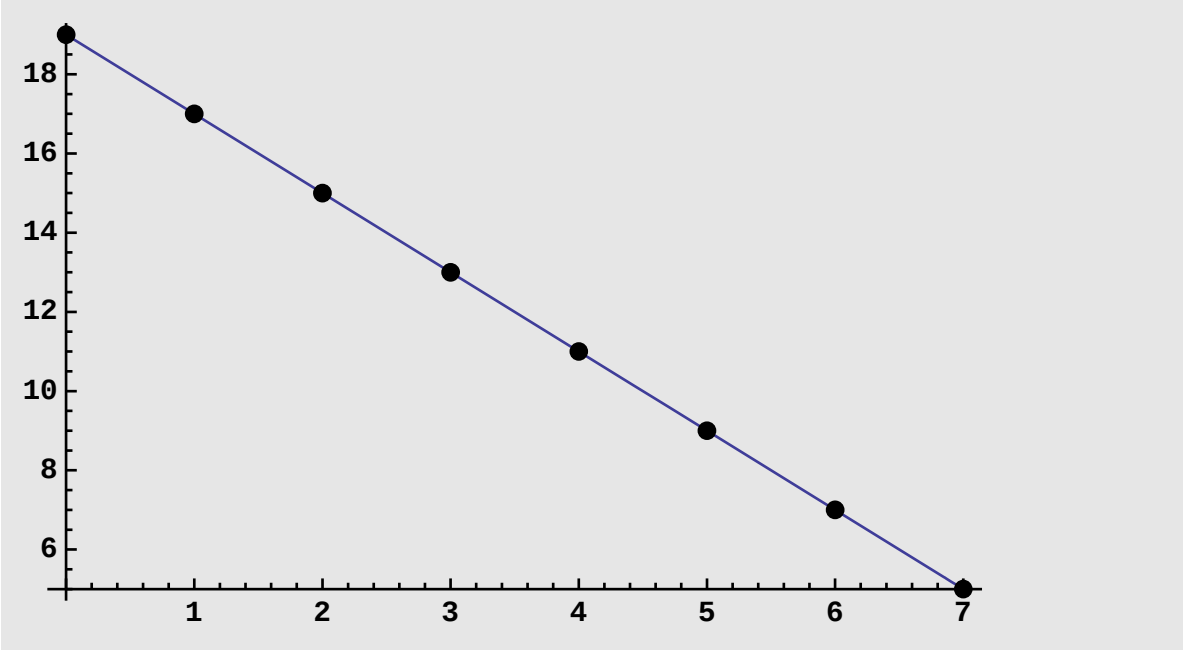
Consider an arbitrary line through the point/secret (0, 19), for example $f(x) = 19 - 2\,x$.

The 7 participants get the shares $(i,\ f(i))$, $i = 1, 2, \ldots, 7$.

```
f[x_] := 19 - 2 x;
{AA, BB, CC, DD, EE, FF, GG} = Table[{i, f[i]}, {i, 1, 7}]
```

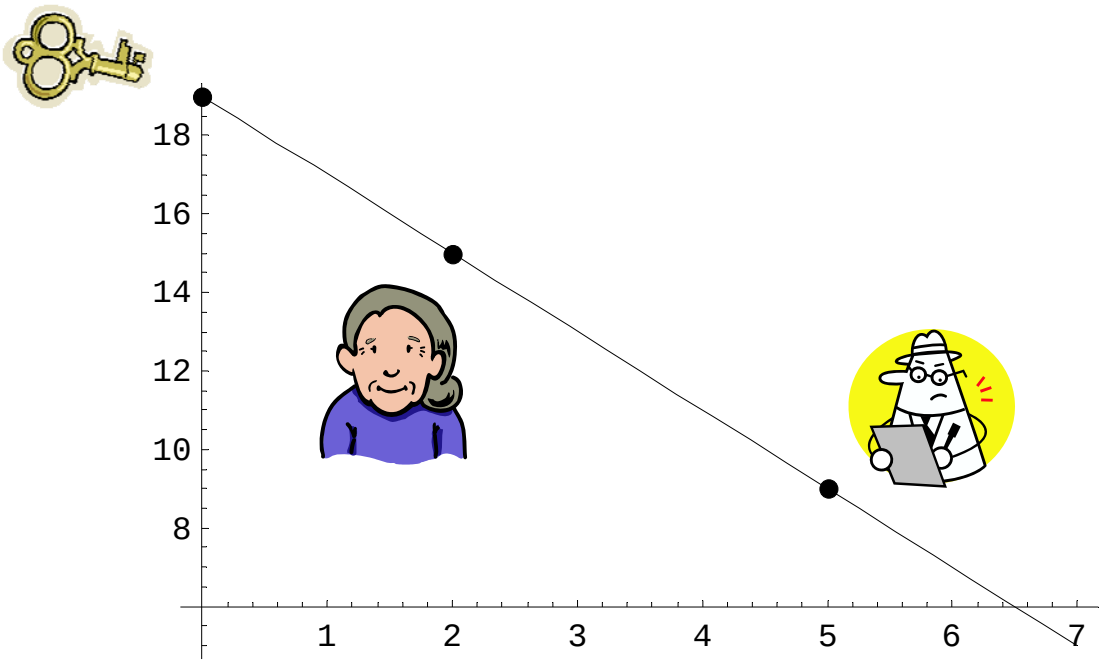
```
{{1, 17}, {2, 15}, {3, 13}, {4, 11}, {5, 9}, {6, 7}, {7, 5}}
```

```
Plot[f[x], {x, 0, 7},
 Epilog -> {PointSize[0.02], Point /@ Table[{i, f[i]}, {i, 0, 7}]}]
```



□ Two participants come together

When 2 participants come together, for example *B* and *E*, they know the points (2, 15) and (5, 9) and can easily find the line $y = u\,x + v$ through them.



They can use the formula

$$y = \frac{y_2 - y_1}{x_2 - x_1}\,(x - x_1) + y_1$$

or solve the two equations with two unknowns

$$15 = 2u + v$$

$$9 = 5u + v$$

```
Solve[{15 == 2 u + v, 9 == 5 u + v}, {u, v}]
```

```
{{u → -2, v → 19}}
```

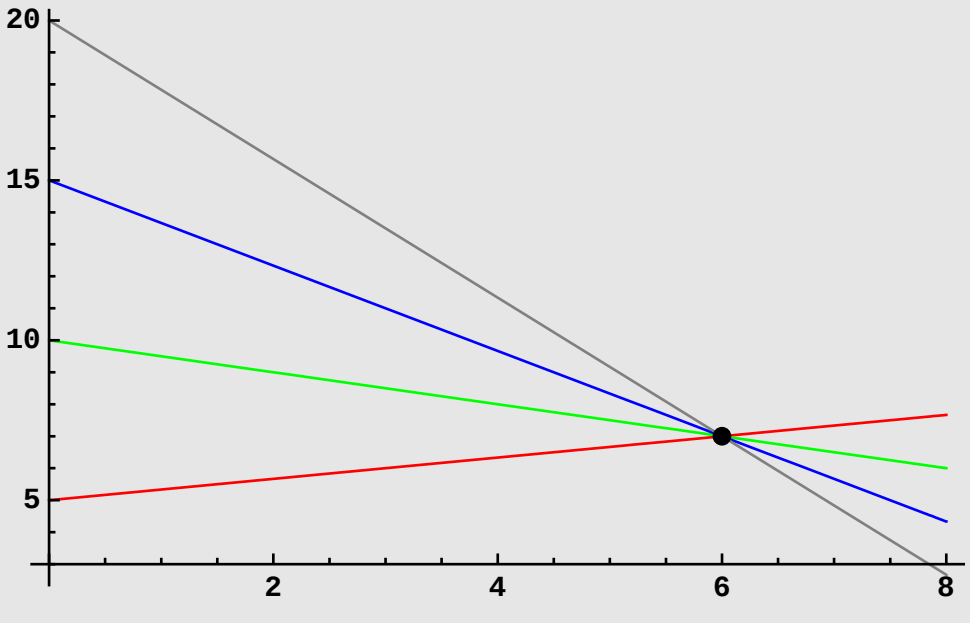
to find the line

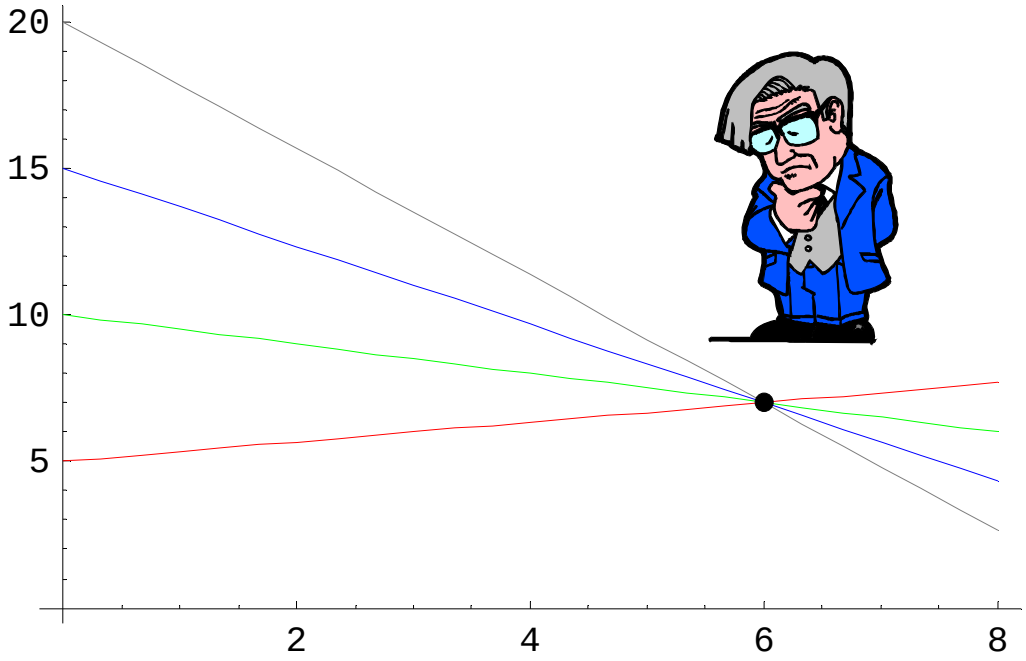
$$y = 19 - 2x$$

The secret follows from substituting $x = 0$, so $S = 19$.

One participant does not know anything. For instance, for F, with share $(6, 7)$, every secret is still equally likely.

```
Null^2
```





□ With liars

Suppose that *E* meets *B*, but purposely gives the wrong value, for example (5, 12). This yields

```
Solve[{15 == 2 u + v, 12 == 5 u + v}, {u, v}]
```

```
{{u → -1, v → 17}}
```

Their calculation gives the line $y = 17 - x$ with "secret" 17. The safe will not open!

Of course, *B* will accuse *E* of fraud, but *E* will accuse *B* equally hard of fraud.

Then *F* enters the room. He has share (6, 7). There are now three candidate lines.

We use "InterpolatingPolynomial", the Lagrange interpolation formula, instead of solving two equations with two unknowns.

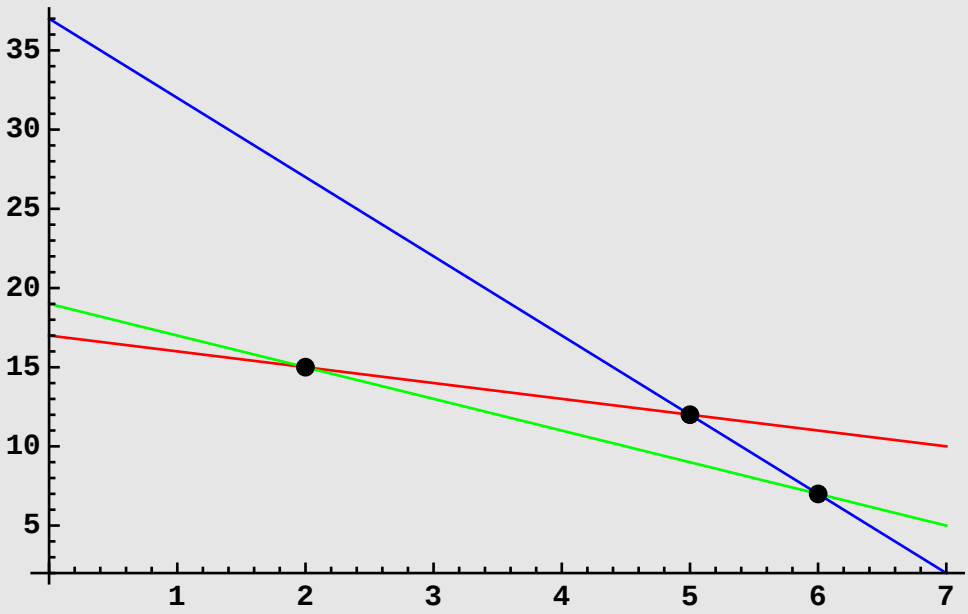
$$y = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1}$$

```
Evil = {5, 12};
g = Expand[InterpolatingPolynomial[{BB, Evil}, x]]
h = Expand[InterpolatingPolynomial[{BB, FF}, x]]
k = Expand[InterpolatingPolynomial[{Evil, FF}, x]]
Plot[{g, h, k}, {x, 0, 7},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0],
   RGBColor[0, 0, 1], RGBColor[0.5`, 0.5`, 0.5`]},
 Epilog → {PointSize[0.02`], Point /@ {BB, Evil, FF}}]
```

```
17 - x
```

```
19 - 2 x
```

```
37 - 5 x
```



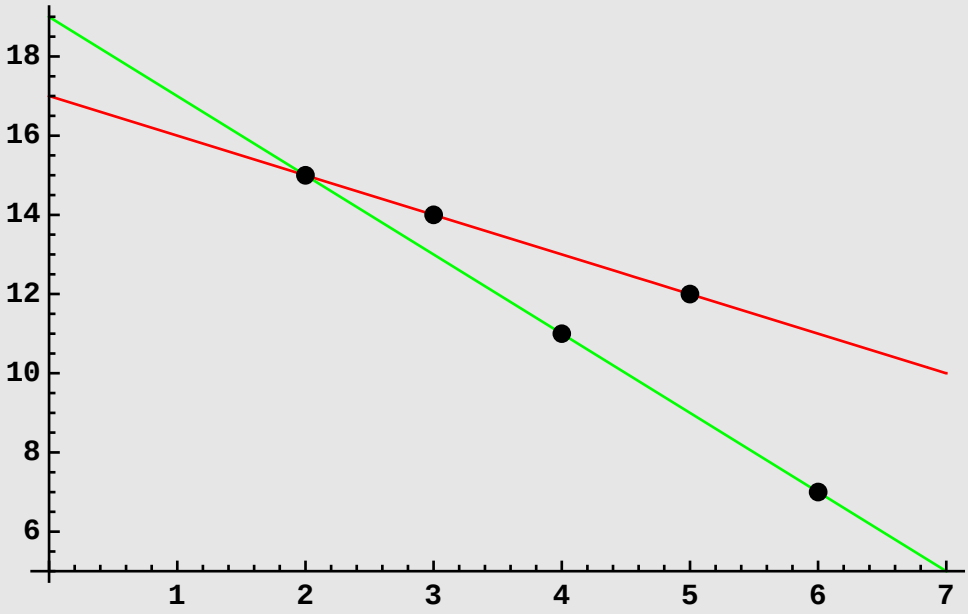
Quite clearly, if one more honest person joins these participants, only the correct (green) line will remain and *E* will be identified as a liar.

Lemma: Let 3 of four given points lie on a line *l*. Then there cannot be a different line, say *m*, also containing 3 of the given points.

Proof: Two 3-tuples from a set of size 4, must have at least 2 points in common. These 2 points already determine *l* and *m*. Apparently *l* = *m*.

If there are 2 liars, 3 honest participants will not always be enough. For instance:

```
Con = {3, 14}; Evil = {5, 12};
g = Expand[InterpolatingPolynomial[{BB, Con, Evil}, x]];
h = Expand[InterpolatingPolynomial[{BB, DD, FF}, x]];
Plot[{g, h}, {x, 0, 7},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0],
   RGBColor[0, 0, 1], RGBColor[0.5`, 0.5`, 0.5`]}, Epilog →
  {PointSize[0.02`], Point /@ {BB, Con, DD, Evil, FF}}]
```



It looks like the two liars have collaborated.

But with six participants, of which at most two are dishonest, there is again no problem.

Lemma: **Let 4 of 6 given points lie on line *l*. Then there cannot be a different line, say *m*, also containing 4 of the given 6 points.**

Proof: Two 4-tuples from a set of size 6, must have at least 2 points in common. These 2 points already determine *l* and *m*. Apparently *l* = *m*.

Etc.

For each liar, you apparently need 2 additional honest participants to resolve the issue!

Theorem: **Consider *2+2e* shares of which at most *e* are false. The remaining shares all lie on the line *l*. Then there cannot be a different line, say *m*, also containing *2+e* of the given points.**

With 1 + 2 *e* shares this is not necessarily the case.

1.4.2 An (7, 3) threshold scheme

□ **Three participants come together**

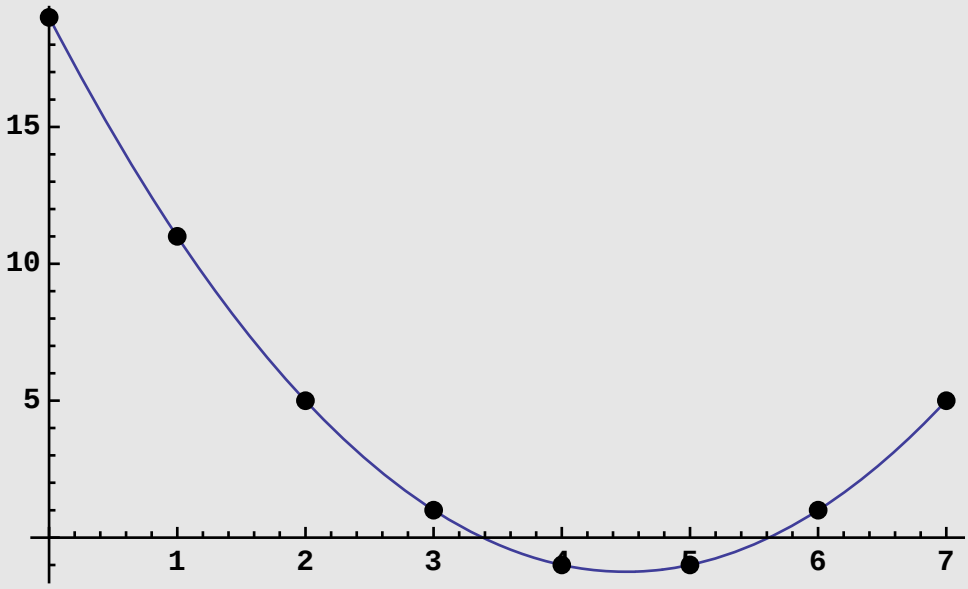
Consider an arbitrary parabola through the point/secret (0, 19), for example $f(x) = 19 - 2x + x^2$.

We give the 7 participants the shares $(i, f(i))$, $i = 1, 2, \ldots, 7$.

```
f[x_] := 19 - 9 x + x^2;
{AA, BB, CC, DD, EE, FF, GG} = Table[{i, f[i]}, {i, 1, 7}]
```

```
{{1, 11}, {2, 5}, {3, 1}, {4, -1}, {5, -1}, {6, 1}, {7, 5}}
```

```
Plot[f[x], {x, 0, 7},
 Epilog -> {PointSize[0.02], Point /@ Table[{i, f[i]}, {i, 0, 7}]}]
```



When 3 participants come together, for example *B* and *E*, they know the points (3, 1), (5, −1) and (6, 1) and can easily find the parabola $y = u x^2 + v x + w$ through them. They solve the three equations with three unknowns

$$1 = u\,3^2 + u\,3 + w$$

$$-1 = u\,5^2 + v\,5 + w$$

$$5 = u\,6^2 + v\,6 + w$$

```
Solve[{1 == u 3^2 + v 3 + w, -1 == u 5^2 + v 5 + w, 1 == u 6^2 + v 6 + w},
 {u, v, w}]
```

```
{{u → 1, v → -9, w → 19}}
```

and find the parabola

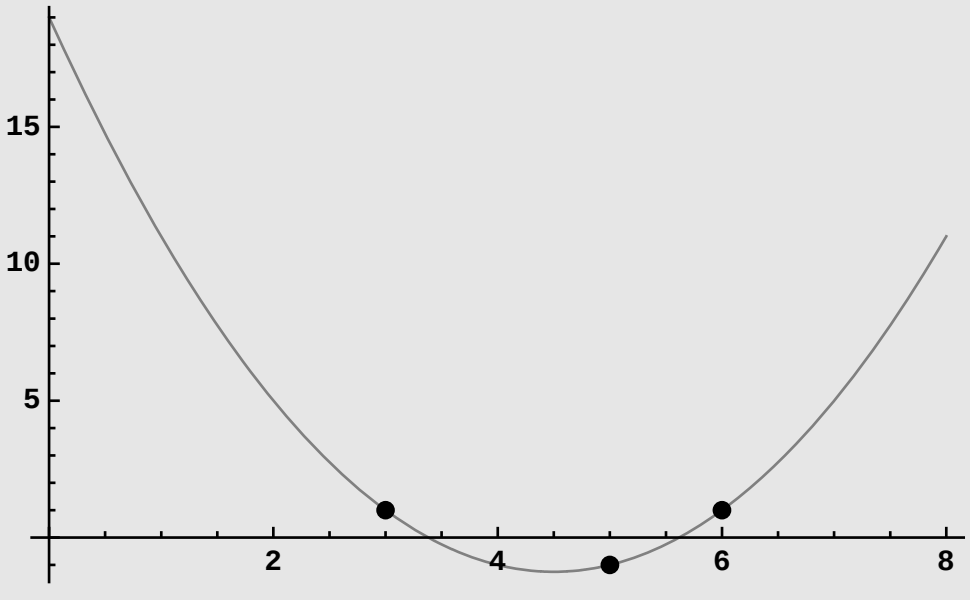
$$x^2 - 9\,x + 19$$

and thus the secret $S = 19$.

Of course, they could have used the *interpolation formula:*

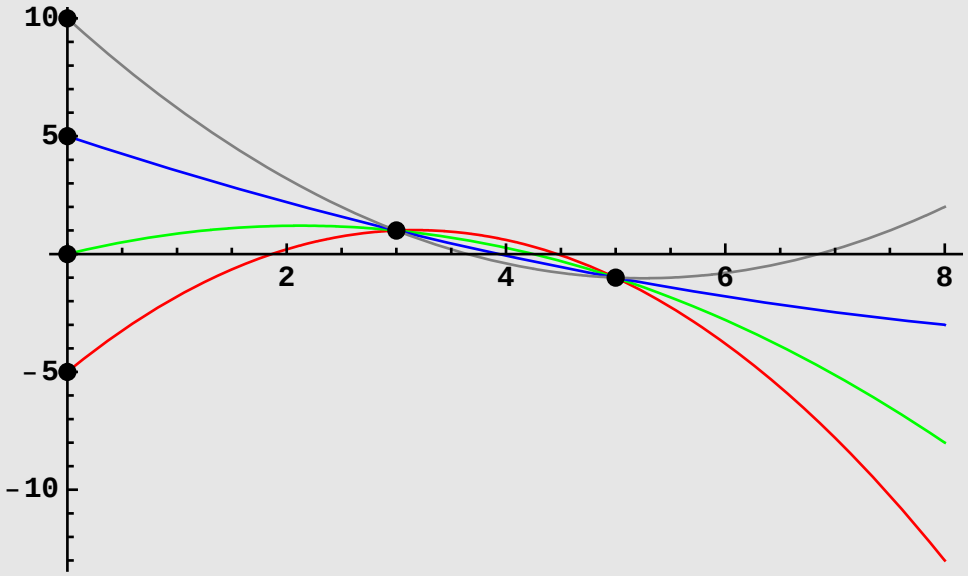
$$y = y_1 \frac{x - x_2}{x_1 - x_2}\frac{x - x_3}{x_1 - x_3} + y_2 \frac{x - x_1}{x_2 - x_1}\frac{x - x_3}{x_2 - x_3} + y_3 \frac{x - x_1}{x_3 - x_1}\frac{x - x_2}{x_3 - x_2}$$

```
g = Expand[InterpolatingPolynomial[{CC, EE, FF}, x]];
Plot[{g}, {x, 0, 8},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0],
   RGBColor[0, 0, 1], RGBColor[0.5`, 0.5`, 0.5`]},
 Epilog → {PointSize[0.02`], Point /@ {CC, EE, FF}}]
```



For two participants every secret is still equally likely. For instance for *C* and *E*

```
g = Expand[InterpolatingPolynomial[{{0, -5}, CC, EE}, x]];
h = Expand[InterpolatingPolynomial[{{0, 0}, CC, EE}, x]];
k = Expand[InterpolatingPolynomial[{{0, 5}, CC, EE}, x]];
l = Expand[InterpolatingPolynomial[{{0, 10}, CC, EE}, x]];
S1 = {0, -5}; S2 = {0, 0}; S3 = {0, 5}; S4 = {0, 10};
Plot[{g, h, k, l}, {x, 0, 8},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0],
   RGBColor[0, 0, 1], RGBColor[0.5`, 0.5`, 0.5`]}, Epilog →
  {PointSize[0.02`], Point /@ {CC, EE, S1, S2, S3, S4}}]
```



□ With liars

Suppose that *E* gives an incorrect share, say (5, 7), in the presence of *C* and *F*. They will solve

```
Solve[
 {1 == u 3^2 + v 3 + w, 7 == u 5^2 + v 5 + w, 1 == u 6^2 + v 6 + w}, {u, v, w}]
```

```
{{u → -3, v → 27, w → -53}}
```

and get the parabola $y = -3\,x^2 + 27\,x - 53$ and thus the secret -53. The safe will not open.

Now *C* knows that *E* or *F* is dishonest (or both), but not which one.

Suppose that *D* joins them with her share (6, 7). There are now 4 candidate parabolas.

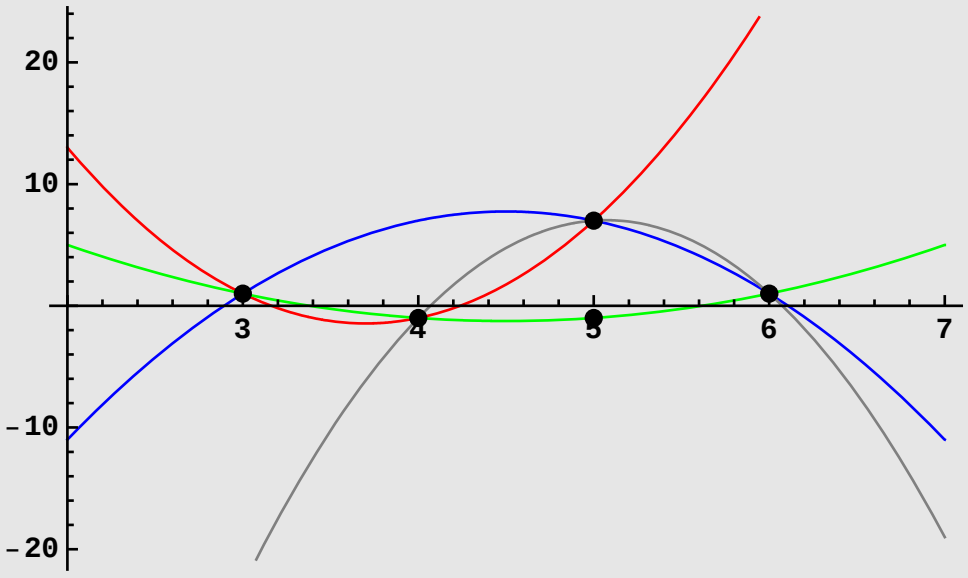
```
Evil = {5, 7};
g = Expand[InterpolatingPolynomial[{CC, DD, Evil}, x]]
h = Expand[InterpolatingPolynomial[{CC, DD, FF}, x]]
k = Expand[InterpolatingPolynomial[{CC, Evil, FF}, x]]
l = Expand[InterpolatingPolynomial[{DD, Evil, FF}, x]]
Plot[{g, h, k, l}, {x, 2, 7},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0],
   RGBColor[0, 0, 1], RGBColor[0.5`, 0.5`, 0.5`]},
 Epilog → {PointSize[0.02`], Point /@ {CC, DD, EE, Evil, FF}}]
```

$$67 - 37\,x + 5\,x^2$$

$$19 - 9\,x + x^2$$

$$-53 + 27\,x - 3\,x^2$$

$$-173 + 71\,x - 7\,x^2$$



Quite clearly, if one more honest person joins these participants, only the correct (green) line will remain and *E* will be identified as a liar.

Lemma: Let 4 of 5 given points lie on parabola *l*. Then there cannot be a different parabola, say *m*, also containing 4 of the given 5 points.

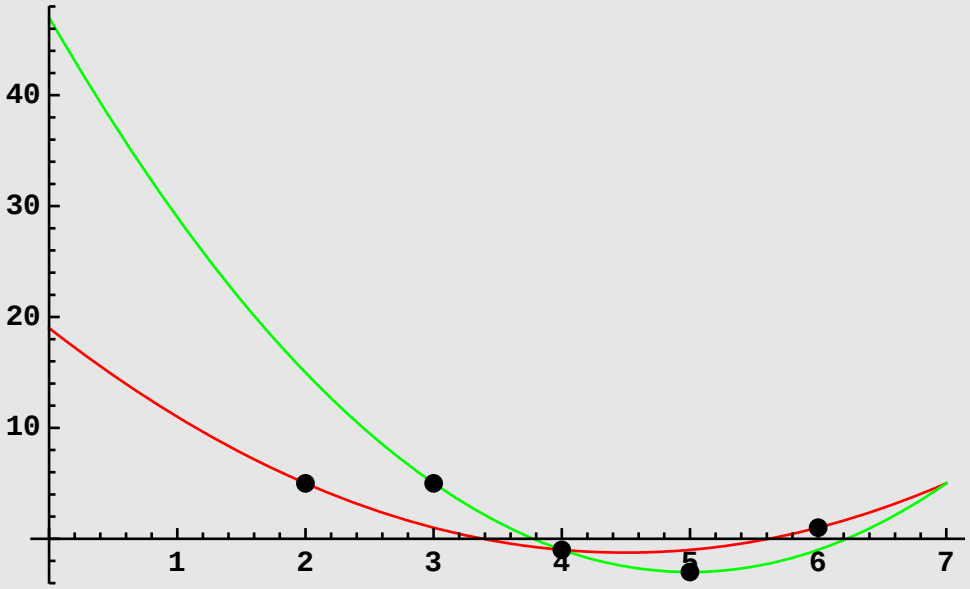
Proof: Two 4-tuples from a set of size 5, must have at least 3 points in common. These 3 points already determine *l* and *m*. Apparently *l* = *m*.

If there are two liars, six participants are not always enough:

```
Con = {3, 5}; Eve = {5, -3};
g = Expand[InterpolatingPolynomial[{CC, DD, EE, FF}, x]]
h = Expand[InterpolatingPolynomial[{Con, DD, Eve}, x]]
Plot[{g, h}, {x, 0, 7},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0]}, Epilog →
  {PointSize[0.02`], Point /@ {{2, 5}, Con, DD, Eve, FF}}]
```

```
19 - 9 x + x^2
```

```
47 - 20 x + 2 x^2
```



But with seven participants, of which at most two are dishonest, there is again no problem.

Lemma: Let 5 of 7 given points lie on parabola *l*. Then there cannot be a different parabola, say *m*, also containing 5 of the given 7 points.

Proof: Two 5-tuples from a set of size 7, must have at least 3 points in common. These 3 points already determine *l* and *m*. Apparently *l* = *m*.

Etc.

For each liar, you apparently need 2 additional honest participants to resolve the issue!

Theorem: Consider 3+*2e* shares of which at most *e* are false. The remaining shares all lie on a parabola *l*. Then there cannot be a different parabola, say *m*, also containing *2+e* of the given points.

With 2 + 2 *e* shares this is not necessarily the case.

1.4.3 An (7, 4) threshold scheme

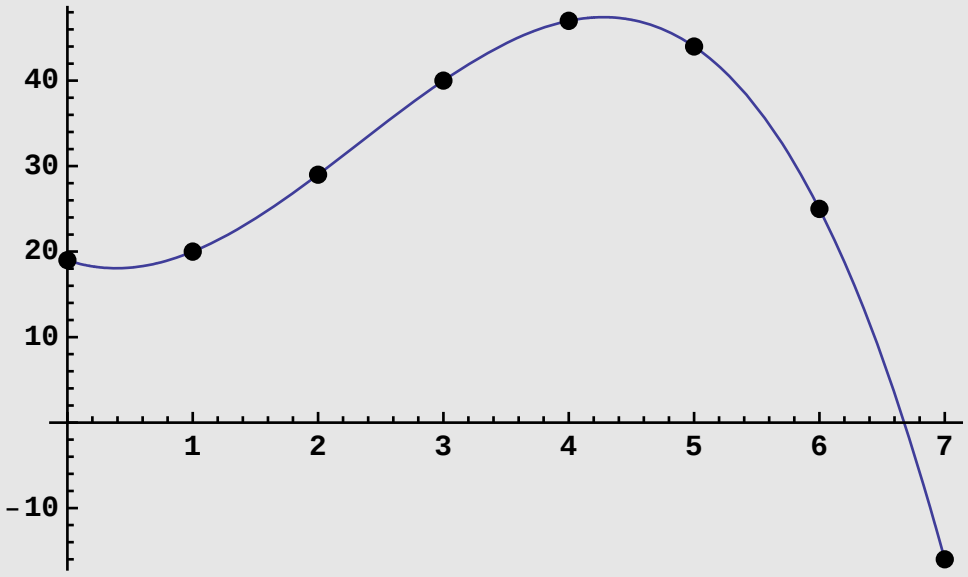
Consider an arbitrary polynomial of degree 3 through the point/secret (0, 19), for example $f(x) = 19 - 5x + 7x^2 - x^3$.

We give the 7 participants the shares $(i, f(i))$, $i = 1, 2, \ldots, 7$.

```
f[x_] := 19 - 5 x + 7 x^2 - x^3;
{AA, BB, CC, DD, EE, FF, GG} = Table[{i, f[i]}, {i, 1, 7}]
```

```
{{1, 20}, {2, 29}, {3, 40}, {4, 47}, {5, 44}, {6, 25}, {7, -16}}
```

```
Plot[f[x], {x, 0, 7},
 Epilog -> {PointSize[0.02], Point /@ Table[{i, f[i]}, {i, 0, 7}]}]
```



When 4 participants come together, for example *A*, *C*, *E* and *F*, they know the points (1, 20), (3, 40), (5, 44) and (6, 25) and can easily find the third degree polynomial $y = u x^3 + v x^2 + w x + t$ through them. They solve the four equations with four unknowns

$$20 = u\,1^3 + u\,1^2 + w\,1 + t$$

$$40 = u\,3^3 + u\,3^2 + w\,3 + t$$

$$44 = u\,5^3 + u\,5^2 + w\,5 + t$$

$$25 = u\,6^3 + u\,6^2 + w\,6 + t$$

```
Solve[{20 == u 1^3 + v 1^2 + w 1 + t, 40 == u 3^3 + v 3^2 + w 3 + t,
   44 == u 5^3 + v 5^2 + w 5 + t, 25 == u 6^3 + v 6^2 + w 6 + t}, {u, v, w, t}]
```

```
{{u → -1, v → 7, w → -5, t → 19}}
```

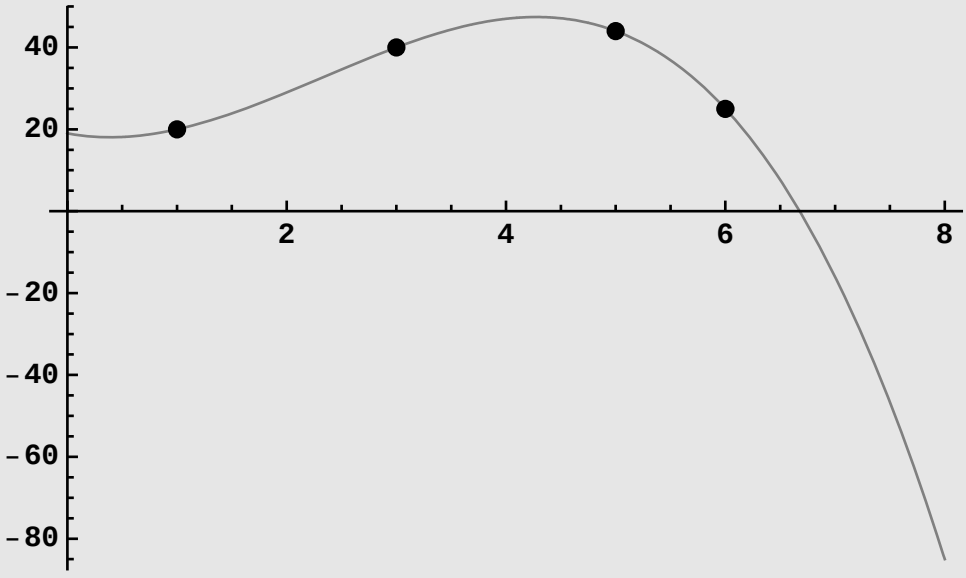
and find the third degree polynomial

$$-x^3 + 7\,x^2 - 5\,x + 19$$

and thus the secret value $S = 19$.

Alternatively, they use

```
g = Expand[InterpolatingPolynomial[{AA, CC, EE, FF}, x]];
Plot[{g}, {x, 0, 8},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0],
   RGBColor[0, 0, 1], RGBColor[0.5`, 0.5`, 0.5`]},
 Epilog → {PointSize[0.02`], Point /@ {AA, CC, EE, FF}}]
```



Etc.

1.4.4 An (*n*, *k*) threshold scheme in general

Theorem:

Take a random polynomial $f(x) = S + a_1 x + \ldots a_{k-1} x^{k-1}$ of degree $k - 1$ (through $(0, S)$).

Participant i, $1 \leq i \leq n$, gets share $(i, f(i))$.

When $\leq (k - 1)$ participants gather,

secret S remains completely unknown (each possibility is equally likely).

k participants can recover $f(x)$ and determine S.

$k + 2$ participants can handle 1 liar.

$k + 4$ participants can handle 2 liars.

Etc.

1.5 The “real” scheme and its connection to RS codes

1.5.1 The (n, k) threshold scheme over a finite field

All the calculations in the previous chapter were over the reals. That is not realistic, because you also get negative numbers and fractions that way and they have no meaning in our context.

More importantly, we like to make statements like "every secret is still equally likely", but we have no uniform probability distribution over the reals.

The solution will be to work over a finite field $\mathbb{F}_q = GF(q)$ of size q.

Construction: of an (n, k)-threshold scheme for secret S in finite field $\mathbb{F}_q$.

Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be n different, non-zero elements in $\mathbb{F}_q$.

Take a random polynomial $f(x) = S + a_1 x + \ldots + a_{k-1} x^{k-1}$ of degree $k - 1$ (through (0, S)),

by choosing the coefficients a_i, $1 \le i \le k - 1$, randomly from GF(q).

Participant i, $1 \le i \le n$, gets share $(\alpha_i, f(\alpha_i))$.

When $\le (k - 1)$ participants gather,

secret S remains completely unknown (each value in GF(q) is equally likely).

k participants can recover $f(x)$ and determine S.

$k + 2$ participants can handle 1 liar.

$k + 4$ participants can handle 2 liars.

Etc.

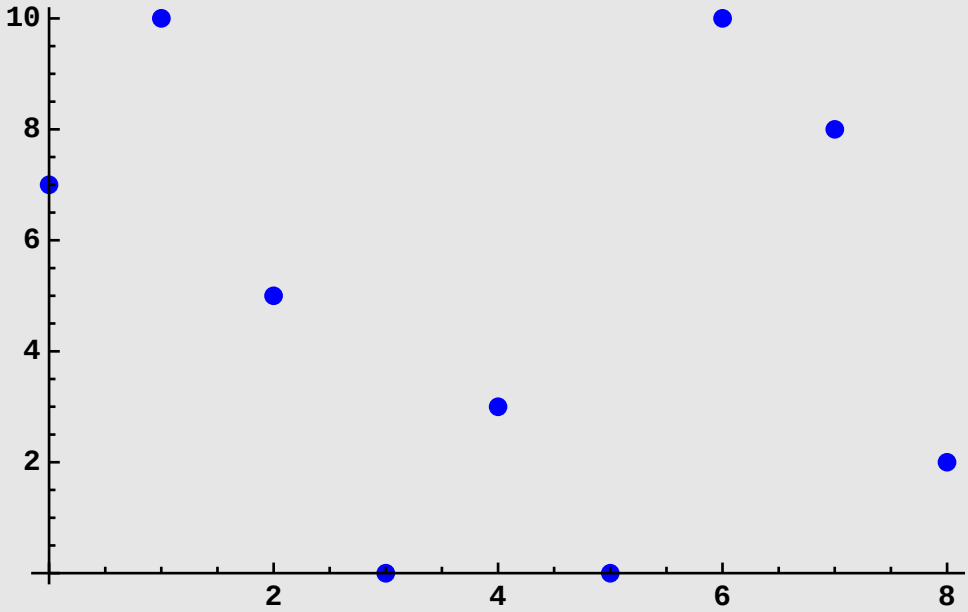
Example: an $(n, k) = (8, 4)$ threshold scheme for a secret in $\mathbb{Z}_{11}$.

We take $\alpha_i = i$, $1 \le i \le 8$.

Let $S = 7$ be the secret and hidden in the polynomial of degree $k = 3$:

```
p = 11;
s = 7;
a = RandomInteger[11];
b = RandomInteger[11]; c = RandomInteger[11];
f[x_] := 7 + a x + b x^2 + c x^3; n = 8;
{AA, BB, CC, DD, EE, FF, GG, HH} =
 Table[{i, Mod[f[i], p]}, {i, 1, n}]
ListPlot[{{0, s}, AA, BB, CC, DD, EE, FF, GG, HH},
 PlotStyle → Directive[PointSize[.02], Blue]]
```

```
{{1, 10}, {2, 5}, {3, 0}, {4, 3}, {5, 0}, {6, 10}, {7, 8}, {8, 2}}
```



If BB, DD, FF, GG come together, they can recover $f(x)$ and the secret by means of

```
Expand[InterpolatingPolynomial[
  {BB, DD, FF, GG}, x, Modulus → p], Modulus → p]
```

$$7 + 6\,x + 3\,x^2 + 5\,x^3$$

Note also that three people know nothing about the secret, in the sense that every secret is equally likely. For instance, the following p polynomials of degree 3 all go through BB, FF, and GG and they take on each value in $\mathbb{Z}_{11}$ exactly once.

```
Table[Expand[InterpolatingPolynomial[{{0, i}, BB, FF, GG},
    x, Modulus → p], Modulus → p], {i, 0, p - 1}] // TableForm
```

$8\,x + 10\,x^2 + 6\,x^3$
$1 + 3\,x + 9\,x^2 + 9\,x^3$
$2 + 9\,x + 8\,x^2 + x^3$
$3 + 4\,x + 7\,x^2 + 4\,x^3$
$4 + 10\,x + 6\,x^2 + 7\,x^3$
$5 + 5\,x + 5\,x^2 + 10\,x^3$
$6 + 4\,x^2 + 2\,x^3$
$7 + 6\,x + 3\,x^2 + 5\,x^3$
$8 + x + 2\,x^2 + 8\,x^3$
$9 + 7\,x + x^2$
$10 + 2\,x + 3\,x^3$

1.5.2 Dealing with liars; the coding theory approach

The way we dealt with liars in Section 4 is of course not practical. With e liars, we need $k + 2e$ shares and have to find $k + e$ honest shares among them.

But the description of the threshold scheme above calls for a coding theory approach.

Definition: Consider the finite field $\mathbb{F}_q = GF(q)$ and put $n = q - 1$.

Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ represent the non-zero elements in $\mathbb{F}_q$. For any $1 \le k \le n$, we define

$$RS(n, k) = \{(f(\alpha_1), f((\alpha_2), \ldots, f(\alpha_n)) \mid f \in \mathbb{F}_q[x], \text{ degree}(f) < k\}.$$

Then RS (n, k) is a linear code of length n over $\mathbb{F}_q$ with minimum distance $d = n - k + 1$

and (is equivalent to) the q-ary Reed-Solomon code of dimension k.

Proof: For $f, g \in \mathbb{F}_q[x]$ and $u, v \in \mathbb{F}_q$ one has $u.f + v.g \in \mathbb{F}_q[x]$ and

$$\text{degree}(f) < k \text{ and } \text{degree}(g) < k \implies \text{degree}(u.f + v.g) < k$$

It follows that RS(n, k) is a linear code of length n.

Clearly, the dimension is equal to k.

A non-zero polynomial $f \in \mathbb{F}_q[x]$ has at most as many zeroes as its degree.

It follows that for $f \neq 0$, the weight of $(f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n))$ is at least $n - (k - 1)$.

As the minimum distance of a linear code equals its minimum non-zero weight, we conclude that $d \geqslant n - k + 1$.

In view of the Singleton bound, equality must hold.

To verify that this code is equivalent to the Reed-Solomon code, take $\alpha_i = \alpha^{i-1}$ for some primitive element α in $\mathbb{F}_q$.

Then RS(n, k) as defined above can be generated by the polynomials x^i, $0 \leqslant i < k$, and thus has generator matrix:

$$G = \begin{pmatrix} 1 & 1 & 1 & \cdots & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & & & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \cdots & \cdots & \alpha^{(k-1)(n-1)} \end{pmatrix},$$

the familiar form of the parity check matrix of the cyclic code (remember that $\alpha^n = \alpha^{q-1} = 1$) with generator polynomial $(x-1)(x-\alpha)\ldots\left(x-\alpha^{k-1}\right)$.

We conclude that G is generator matrix of a cyclic code with $h(x) = (x-1)(x-\alpha)\ldots\left(x-\alpha^{k-1}\right)$ as parity check polynomial and thus with $g(x) = \left(x-\alpha^k\right)\left(x-\alpha^{k+1}\right)\ldots\left(x-\alpha^{n-1}\right) = \left(x-\alpha^{-1}\right)\left(x-\alpha^{-2}\right)\ldots\left(x-\alpha^{-(n-k)}\right)$ as generator polynomial.

In view of the BCH bound this is another way of seeing that RS(n, k) is an $[n, k, n-k+1]_q$ code.

The general construction of an (n, k) -threshold scheme in $1.5.1 allows for $n < q-1$. In that case, some field elements are not substituted in the polynomial f.

The code that one now obtains is called a *shortened* Reed-Solomon code.

If the threshold scheme must be able to handle e liars, the corresponding RS code must be able to correct e errors, so its minimum distance must be at least $2e+1$.

The redundancy of this code satisfies $r = n-k = d-1 = 2e$.

This corresponds to the $2e$ additional honest participants that were needed in the threshold scheme.

The good news for threshold schemes is that (shortened) Reed-Solomon codes allow very efficient decoding (Peterson, Berlekamp, Welch, etc.).

Singleton bound: Let C be a code of length n with minimum distance d. Then

$$|C| \leq q^{n-d+1}$$

In the special case that C is a linear $[n, k, d]$ code, and thus $|C| = q^k$, it follows that

$$d \leq n-k+1.$$

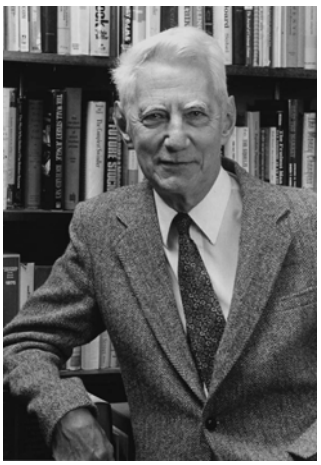
Codes that meet this bound with equality are called "optimal".

It follows that Reed-Solomon codes are optimal.

2 Symmetric Systems

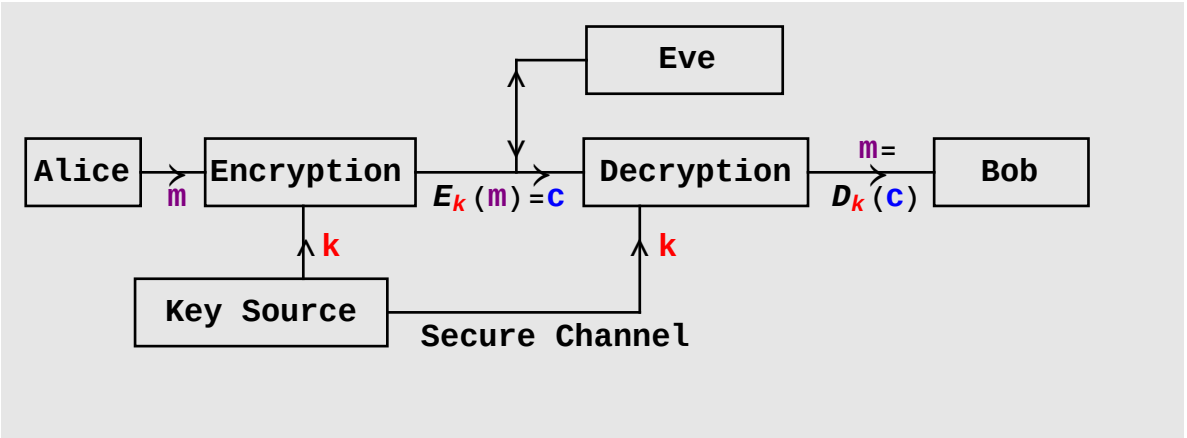
Symmetric cryptosystems have been used for at least 2000 years. In these systems, the sender of a message and the intended receiver must share a common key.

They are very popular because of their "symplicity" and high speed of operation. They are not based on mathematical assumptions, but make use of the ingredients that Claude Shannon recommended back in 1947: "confusion" and "diffusion".



Wikipedia*: Claude E. Shannon* (April 30, 1916 – February 24, 2001) was an American mathematician, electronic engineer, and cryptographer known as "the father of information theory".

Shannon's description of a symmetric key cryptosystem is depicted in the following figure.



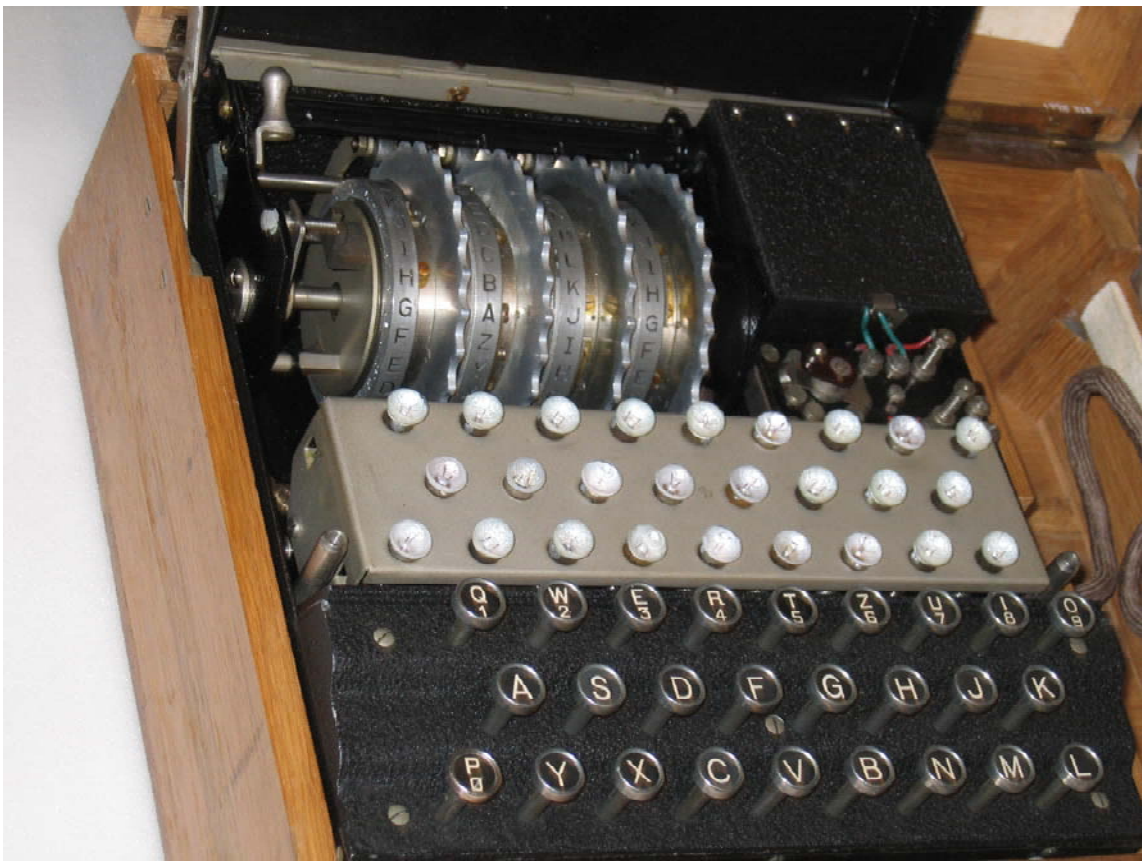
Here, E_k is a 1-1 mapping, *encrypting* plaintext m (clear text) into ciphertext c.

D_k is the inverse mapping, called *decryption*.

The encryption depends on a secret key *k* that sender and receiver share.

One must assume that the encryption method is known to the eavesdropper Eve, but not the actual choice of the key *k*.

□ **Enigma**



The key of the Enigma consists of

i) the choice and order of the rotors,
ii) their initial position and
iii) a fixed initial permutation of the alphabet.

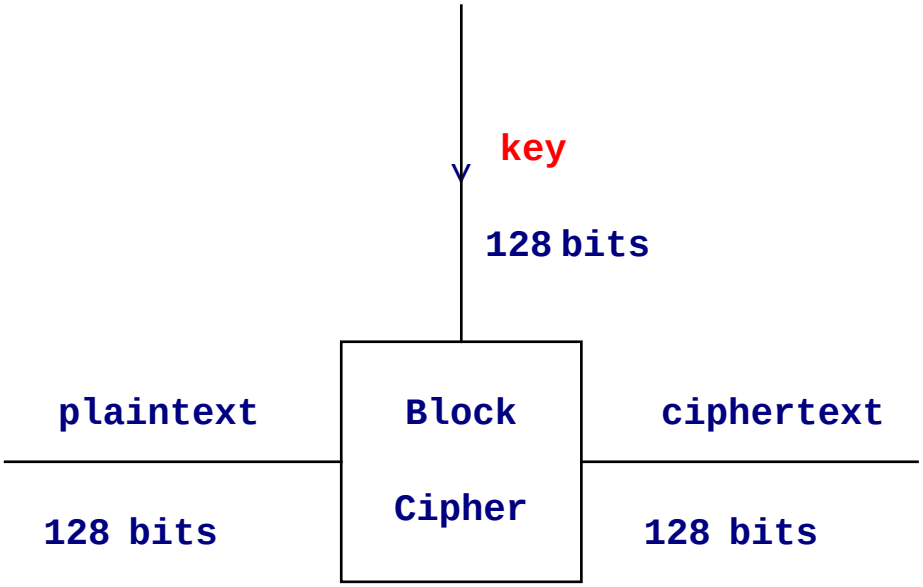
3 Block Ciphers

3.1 Introduction

Block ciphers handle *n* bits at a time. Typical values are *n* = 64, 128, 192, and 256.

They have no memory and can operate at very high speeds.

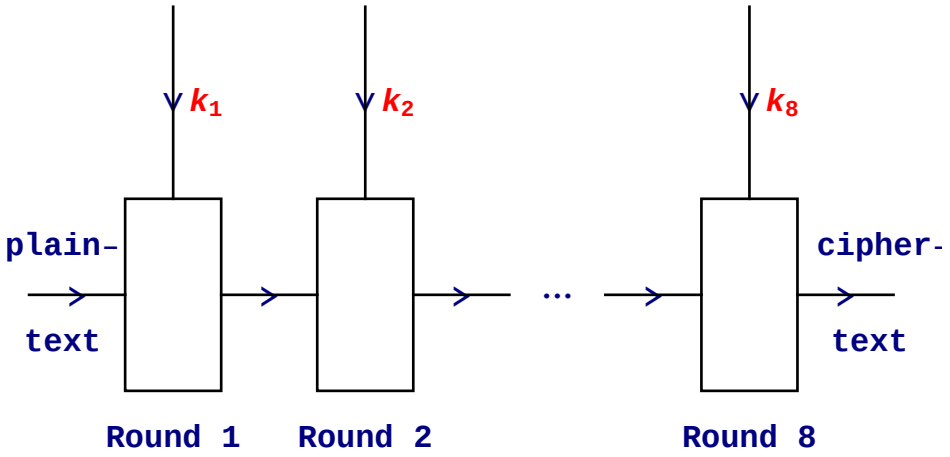
□ Electronic Codebook



Often, the same device can be used for encryption and decryption.

Typically, the block cipher consists of a sequence of identical looking rounds each operating under a *round key* that is computed from the key k.

Each round is designed to realize "confusion" and "diffusion" in order to obscure dependencies and other statistical properties of the plaintext.



Note that the same plaintext will result in the same ciphertext as long as the key has not been changed. To avoid this situation feedback is introduced.

□ **DES**

The Data Encryption Standard (DES) is such a system. It operates on 64 bits at a time.

It became a standard in 1976. Its (effective) key size is 56 bits.

In 1999 a large collection of computers broke DES for the first time by *exhaustive key search*.

3.2 Advanced Encryption Standard (AES); Rijndael

3.2.1 AES

At the end of last century, the (American) National Institute of Standards and Technology (NIST) invited the cryptographic community to develop a successor for DES .

The names of these proposals are CAST-256, CRYPTON, DFC, DEAL, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, RIJNDAAEL, SAFER+, SERPENT and TWOFISH.

The second round of the selection was concluded in August 1999. The following contenders remained in the race: MARS (IBM), RC6TM (RSA Laboratories), RIJNDAEL (Daemen and Rijmen), SERPENT (Ross Anderson, Eli Biham, and Lars Knudsen) and TWOFISH (Bruce Schneier e.a.).

On October 2, 2000, the final selection was made: RIJNDAEL!

The block length and the key length in Rijndael can be independently specified to 128, 192, and 256 bits.

The number of rounds in Rijndael depends in the following way on the the block length and the key length.

cipher\key	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

The round function consists of the following operations:

- **ByteSub (affects individual bytes),**
- **ShiftRow (shifts rows),**
- **MixColumn (affects each column),**
- **RoundKey addition (overall XOR).**

These are applied to the intermediate cipher result, also called the State: a 4×4, 4×6, resp. 4×8 matrix of which the entries consist of 8 bits, i.e. one byte. For example, when the block length is 192, one gets

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

where each $a_{i,j}$ consists of 8 bits. For example, $a_{0,0} = \{1, 0, 1, 1, 0, 0, 0, 1\}$.

Let N_b be the number of columns in the array above. So, the the block cipher length is 32 N_b bits, or 4 N_b bytes (each byte consists of 8 bits), or N_b 4-byte words.

3.2.2 One Round of Rijndael

□ **ByteSub**

This is the only non-linear part in each round.

Apply to each byte $a_{i,j}$ two operations:

1) Interpret $a_{i,j}$ as element in $\mathbb{F}_{256} = \mathrm{GF}(2^8)$ and replace it by its multiplicative inverse, if it is not 0, otherwise leave it the same.

2) Replace the resulting 8-tuple, say $(x_0, x_1, \ldots, x_7)$ by

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

The finite field $\mathrm{GF}(2^8)$ is made by means of the irreducible polynomial $m(\alpha) = 1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8$. This polynomial is not primitive!

Note that both operations are invertible.

```
<<FiniteFields`
```

```
f256 = GF[2, {1, 1, 0, 1, 1, 0, 0, 0, 1}];
one = f256[{1, 0, 0, 0, 0, 0, 0, 0}]
α = f256[{0, 1, 0, 0, 0, 0, 0, 0}]
```

$$\{1, 0, 0, 0, 0, 0, 0, 0\}_2$$

$$\{0, 1, 0, 0, 0, 0, 0, 0\}_2$$

```
in = {0, 1, 0, 0, 0, 0, 0, 0};
pol = Sum[in[[i]] α^(i-1), {i, 1, 8}]
inver = 1 / pol
```

$$\{0, 1, 0, 0, 0, 0, 0, 0\}_2$$

$$\{1, 0, 1, 1, 0, 0, 0, 1\}_2$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix};$$

```
b = {1, 1, 0, 0, 0, 1, 1, 0};
Mod[A.inver[[1]] + b, 2]
```

$$\{1, 1, 1, 0, 1, 1, 1, 0\}$$

Instead of performing these calculations, one can also replace them by one substitution table: the ByteSub S-box.

□ ShiftRow

The rows of the State are shifted cyclically to the left using different offsets: do not shift row 0, shift row 1 over c_1 bytes, row 2 over c_2 bytes, and row 3 over c_3 bytes, where

	c_1	c_2	c_3
128	1	2	3
192	1	2	3
256	1	3	4

.

So

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

becomes

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

□ MixColumn

Interpret each column as a polynomial of degree 3 in x over $\mathrm{GF}(2^8)$ and multiply it with

$$(1+\alpha)\,x^3+x^2+x+\alpha$$

modulo x^4+1.

Note that the above polynomial is invertible modulo x^4+1.

```
g = .;
g[x_] = (1 + α) x^3 + one x^2 + one x + α
```

```
{0, 1, 0, 0, 0, 0, 0, 0}2 + x {1, 0, 0, 0, 0, 0, 0, 0}2 +
 x^2 {1, 0, 0, 0, 0, 0, 0, 0}2 + x^3 {1, 1, 0, 0, 0, 0, 0, 0}2
```

Suppose that the first column looks like

```
col = {1 + α + α^3 + α^6 + α^7, one, α^2 + α^4 + α^5 + α^6, α};
col // TableForm
```

```
{1, 1, 0, 1, 0, 0, 1, 1}2
{1, 0, 0, 0, 0, 0, 0, 0}2
{0, 0, 1, 0, 1, 1, 1, 0}2
{0, 1, 0, 0, 0, 0, 0, 0}2
```

```
colpol[x_] = col[[1]] + col[[2]] x + col[[3]] x^2 + col[[4]] x^3
```

```
x^2 {0, 0, 1, 0, 1, 1, 1, 0}_2 + x^3 {0, 1, 0, 0, 0, 0, 0, 0}_2 +
 x {1, 0, 0, 0, 0, 0, 0, 0}_2 + {1, 1, 0, 1, 0, 0, 1, 1}_2
```

```
ownexpand[expr_] :=
 Collect[expr /. {GF → GF$}, x] /. {GF$ → GF}
```

```
pr[x_] = ownexpand[colpol[x] * g[x]]
prod[x_] = PolynomialMod[pr[x], x^4 - 1]
```

```
x^2 {0, 1, 0, 0, 0, 1, 0, 0}_2 +
 x^6 {0, 1, 1, 0, 0, 0, 0, 0}_2 + x^5 {0, 1, 1, 1, 1, 0, 0, 1}_2 +
 x {1, 0, 0, 1, 0, 0, 1, 1}_2 + x^4 {1, 0, 1, 0, 1, 1, 1, 0}_2 +
 {1, 0, 1, 1, 0, 0, 0, 1}_2 + x^3 {1, 1, 1, 0, 1, 1, 0, 0}_2
```

```
{0, 0, 0, 1, 1, 1, 1, 1}_2 + x^2 {0, 0, 1, 0, 0, 1, 0, 0}_2 +
 x {1, 1, 1, 0, 1, 0, 1, 0}_2 + x^3 {1, 1, 1, 0, 1, 1, 0, 0}_2
```

```
colafter = CoefficientList[prod[x], x];
colafter // TableForm
```

```
{0, 0, 0, 1, 1, 1, 1, 1}_2
{1, 1, 1, 0, 1, 0, 1, 0}_2
{0, 0, 1, 0, 0, 1, 0, 0}_2
{1, 1, 1, 0, 1, 1, 0, 0}_2
```

The inverse operation is a multiplication by

```
h = .; x = .;
h[x_] = (1 + α + α^3) x^3 + (1 + α^2 + α^3) x^2 + (1 + α^3) x + (α + α^2 + α^3) ;
ownexpand[PolynomialMod[g[x] * h[x], x^4 - 1]]
```

```
{1, 0, 0, 0, 0, 0, 0, 0}_2
```

```
ownexpand[PolynomialMod[prod[x] * h[x], x^4 - 1]]
```

```
x^2 {0, 0, 1, 0, 1, 1, 1, 0}_2 + x^3 {0, 1, 0, 0, 0, 0, 0, 0}_2 +
 x {1, 0, 0, 0, 0, 0, 0, 0}_2 + {1, 1, 0, 1, 0, 0, 1, 1}_2
```

These design choices lead to the following properties:

- **Each column of (Hamming) weight 1 is mapped to a column of weight 4.**
- **Each column of weight 2 is mapped to a column of weight at least 3.**
- **Each column of weight 3 is mapped to a column of weight at least 2.**
- **Each column of weight 4 is mapped to a column of weight at least 1.**

Now the connection with Coding Theory

Make "codewords" of length 8, by attaching the column after this MixColumn operation to the column before the operation in all possible ways.

```
Join[col, colafter]
```

```
{{1, 1, 0, 1, 0, 0, 1, 1}2, {1, 0, 0, 0, 0, 0, 0, 0}2,
 {0, 0, 1, 0, 1, 1, 1, 0}2, {0, 1, 0, 0, 0, 0, 0, 0}2,
 {0, 0, 0, 1, 1, 1, 1, 1}2, {1, 1, 1, 0, 1, 0, 1, 0}2,
 {0, 0, 1, 0, 0, 1, 0, 0}2, {1, 1, 1, 0, 1, 1, 0, 0}2}
```

This code is linear and has dimension 4.

One apparently obtains an $[8, 4, 5]_{256}$ code!

The designers could have chosen a shortened Reed-Solomon code for this purpose, starting with an $[255, 251, 5]_{256}$ RS code and shortening it to 8 coordinates.

But they checked all $[8, 4, 5]_{256}$ codes and took the most suitable for them, among others maximizing the number of 1's in the generator matrix (Vincent Rijmen, private communication 2014).

□ Round Key Addition

XOR the whole matrix with a similar sized matrix (i.e. the Round Key) obtained from the cipher key in a way that depends on the round index.

Note that the XOR applied to a byte, really is an XOR applied to the 8 bits in the byte.

For example, if

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$\oplus$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

$$= \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} & u_{0,4} & u_{0,5} \\ u_{1,0} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,5} \\ u_{2,0} & u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & u_{2,5} \\ u_{3,0} & u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} & u_{3,5} \end{pmatrix}.$$

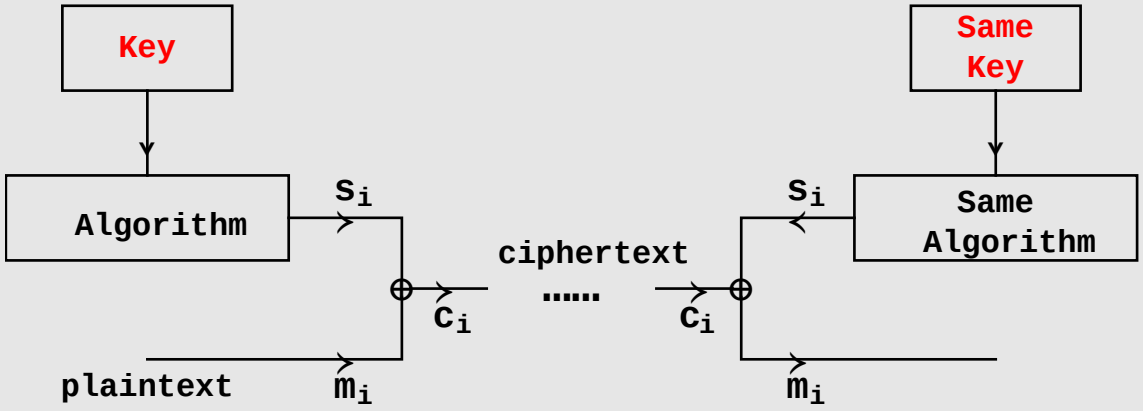
with $u_{0,0} = a_{0,0} \oplus k_{0,0}$, the coordinate-wise exclusive or.

```
a0,0 = {1, 1, 1, 1, 0, 0, 0, 0}; k0,0 = {1, 1, 0, 0, 1, 0, 1, 0};
Mod[a0,0 + k0,0, 2]
```

```
{0, 0, 1, 1, 1, 0, 1, 0}
```

4 Stream Ciphers

4.1 Introduction



A stream cipher outputs one bit at a time.

It has some memory, say *n* bits.

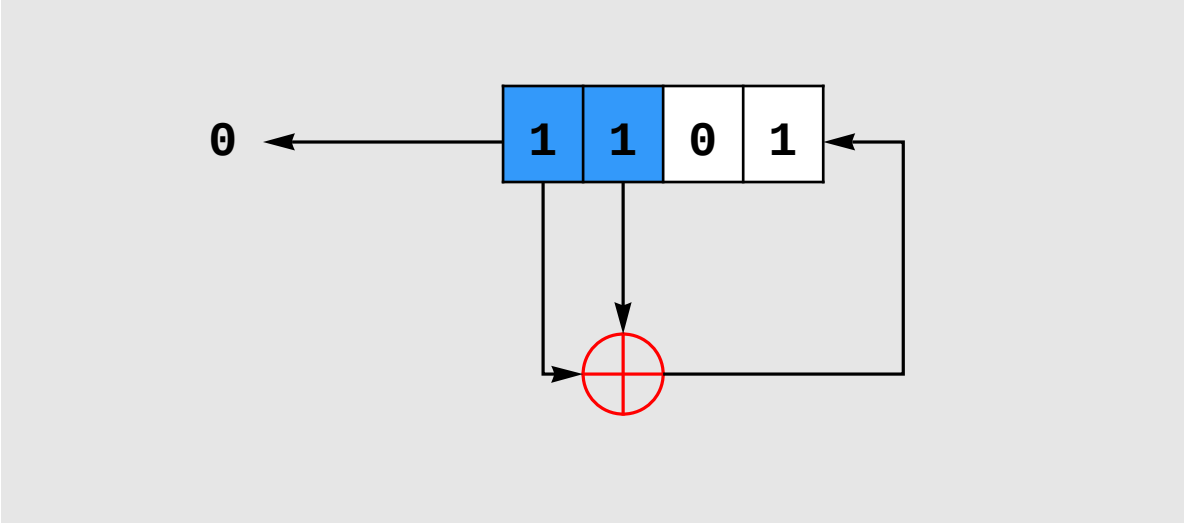
It makes use of very simple electronic circuitry, like shift registers.

It can operate at very high speeds.

4.1.1 Linear Feedback Shift Registers

The easiest building blocks for stream ciphers are Linear Feedback Shift Registers (LFSR).

An example is given by:



Let s_0, s_1, s_2, s_3 be the initial *state* of this register.

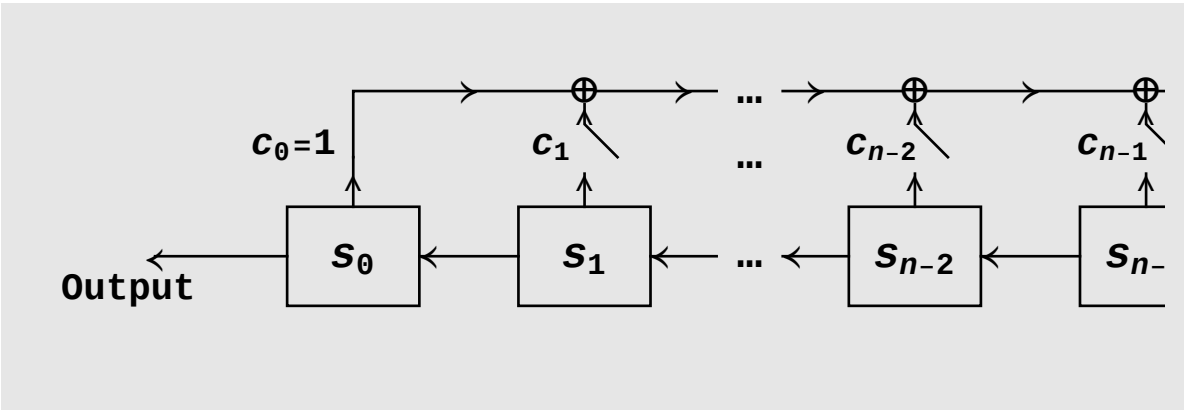
After one clock pulse, s_0 will leave the register as output bit and the state changes to s_1, s_2, s_3, s_4 with $s_4 = s_0 \oplus s_1$.

Then s_1 will leave the register as output bit and the state changes to s_2, s_3, s_4, s_5 with $s_5 = s_1 \oplus s_4$.

So, the output bits s_0, s_1, s_2, ... of the register above satisfy the recurrence relation:

$$s_{k+4} = s_k \oplus s_{k+1}, \qquad k \geq 0.$$

With the general picture of an LFSR



the output sequence $\{s_i\}_{i\geq 0}$ is determined by the initial state $(s_0, s_1, \ldots, s_{n-1})$ and the recurrence relation:

$$s_{k+n} = c_0\, s_k \oplus c_1\, s_{k+1} \oplus \ldots \oplus c_{n-1}\, s_{k+n-1}, \qquad k \geq 0. \tag{4.1}$$

The coefficients c_i, $0 \leq i \leq n-1$ are called the feedback coefficients.

Without proof we mention:

> **Theorem**
> **Let $\{s_i\}_{i\geq 0}$ be the output sequence of (4,1). Then this sequence is periodic with period at most $2^n - 1$.**
>
> **Moreover the period is equal to $2^n - 1$ if and only if the polynomial $c_0 + c_1\, x + \ldots c_{n-1}\, x^{n-1} + x^n$ is a primitive polynomial.**

The polynomial $f(x) = c_0 + c_1\, x + \ldots c_{n-1}\, x^{n-1} + x^n$ is called the characteristic polynomial of the LFSR.

It suffices to know $2\,n$ consecutive output bits of the sequence $\{s_i\}_{i\geq 0}$ to determine the actual LFSR in use and thus to determine the whole sequence.

For instance, if a cryptanalist knows or can guess $s_0, s_1, \ldots, s_{2\,n-1}$, then the feedback coefficients c_i can be computed from the n linear equations with n unknowns

$$\begin{pmatrix} s_k & s_{k+1} & \cdots & \cdots & s_{k+n-1} \\ s_{k+1} & s_{k+2} & \cdots & \cdots & s_{k+n} \\ . & . & \cdots & \cdots & . \\ . & . & \cdots & \cdots & . \\ . & . & \cdots & \cdots & . \\ s_{k+n-1} & s_{k+n} & \cdots & \cdots & s_{k+2\,n-2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ . \\ . \\ . \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} s_{k+n} \\ s_{k+n-1} \\ . \\ . \\ . \\ s_{k+2\,n-1} \end{pmatrix}.$$

For this reason, one can not use LFRS's directly for encryption purposes. One has to combine more LFSR's in a non-linear way.

4.1.2 The A5/1 for GSM

In GSM every conversation consists of sequence of frames, each lasting 4.6 millisecond.

Each frame contains 114 bits for the communication from Alice to Bob and 114 for the communication from Bob to Alice.

Each conversation makes use of a session key K of 64 bits.

For each frame, the session key K and the publicly known frame counter F_n (22 bits long) generate 228 bits that are XOR-ed with the $2 \times 114 = 228$ bits of plaintext.

□ **How are the 228 bits generated?**

Use the three LFSR's depicted below. Their output is XOR-ed to give the output sequence that is XOR-ed with the plaintext.

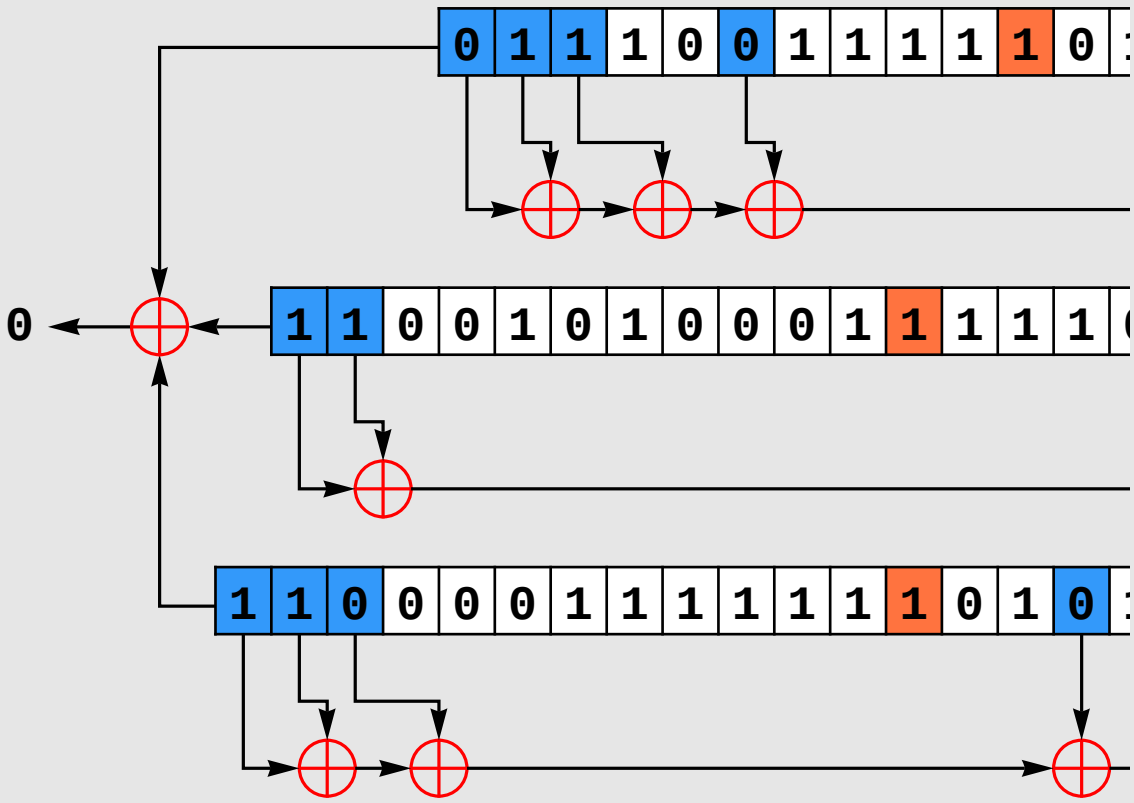
The characteristic polynomials of the registers aren given by

$1 + x + x^2 + x^5 + x^{19}$ **for R1**

$1 + x + x^{22}$	for R2
$1 + x + x^2 + x^{16} + x^{23}$	for R3

However the LFSR's are not always shifted at the same time. Each one has a single "clocking" tap (at positions 8, 10, and 10 for R_1, R_2, resp. R_3). At each clock cycle those registers will shift that agree on their clocking tap with the majority value *m* of the three clocking taps c1, c2 and c3. This is called the"stop/go" rule.

(Note that always at least 2 LFSR's will shift.)



The generation of the 228 bits:

Put the LFRS's in their zero-state. Feed in the 64 bits of *K* by clocking 64 times all three LFRS's (so no stop/go rule), each time XOR-ing the next bit of *K* in parallel with the right most cell of each register .

Continue in exactly the same way 22 more clock cycles to feed in the 22 bits of F_n.

The three LFRS's are clocked 100 more times with the stop/go rule. No output is generated so far.

The three LFRS's are clocked 228 more times with the stop/go rule to generate 228 bits of output.

4.2 The Berlekamp-Massey Algorithm

Let us now consider an output sequence $\{s_i\}_{i\geq 0}$ made in a nonlinear way by a stream cipher, which is a deterministic algorithm in a finite state machine. This sequence will eventually be periodic, say with period p.

This means that, except for a beginning part, $\{s_i\}_{i\geq 0}$ can be generated in a trivial way by the LFSR with characteristic polynomial $1 + x^p$. Indeed, if the period is p then

$$s_{k+p} = s_k, \qquad k \geq 0.$$

Therefore, the sequence $\{s_i\}_{i\geq 0}$ which was possibly made in a non-linear way, can also be made by a LFSR (except for a finite beginning part). Here, we shall assume that the output sequence is periodic right from the start. The discussion above justifies the following definition.

> **Definition**
> **The linear complexity (or linear equivalence) of a periodic sequence $\{s_i\}_{i\geq 0}$ is the length of the smallest LFSR that can generate $\{s_i\}_{i\geq 0}$.**

A cryptanalist, who knows a segment of the output sequence, say $s_0, s_1, \ldots, s_{k-1}$, can try the following strategy:

i) find the smallest LFSR that generates $s_0, s_1, \ldots, s_{k-1}$,
ii) determine the next output bit of this LFSR and hope that it correctly "predicts" the next bit s_k of the sequence.

In that way, the cryptanalist can try to find the characteristic polynomial of the shortest LFSR that can generate $\{s_i\}_{i\geq 0}$.

With the sequence $\{s_i\}_{i\geq 0}$ we associate the power series (also called generating function)

$$S(x) = \sum_{i=0}^{\infty} s_i\, x^i.$$

Theorem 4.2
Each output sequence $\{s_i\}_{i\geq 0}$ of the LFSR with characteristic polynomial $f(x)$ defines a unique polynomial $\omega(x)$ of degree less than n, by means of

$$S(x)\, f^*(x) = \omega(x)\,,$$

where $f^*(x) = c_0\, x^n + c_1\, x^{n-1} + \ldots c_{n-1}\, x + c_n$ is the reciprocal of $f(x)$.

Proof: For each $k \geq 0$, the coeffient of x^{n+k} in

$$S(x)\, f^*(x) = \left(s_0 + s_1\, x + s_2\, x^2 + \ldots\right)\left(c_0\, x^n + c_1\, x^{n-1} + \ldots c_{n-1}\, x + c_n\right)$$

is given by

$$c_0\, s_k + c_1\, s_{k+1} + \ldots + c_{n-1}\, s_{k+n-1} + c_n\, s_{k+n}$$

which is 0 by (4.1).

Also, if $S_1(x)\, f^*(x) = \omega(x)$ and $S_2(x)\, f^*(x) = \omega(x)$, then $(S_1(x) - S_2(x))\, f^*(x) = 0$ implies that $S_1(x) = S_2(x)$.

Of course, we only know $s_0, s_1, \ldots, s_{k-1}$, so we have to find $f^*(x)$ and $\omega(x)$ from

$$S(x)\, f^*(x) = \omega(x) \qquad (\text{mod } x^k).$$

Massey (1969) saw the similarity between the equation $S(x)\, f^*(x) = \omega(x)$ and the so-called key-equation in a decoding algorithm for cyclic codes that Berlekamp presented two years earlier.

Jim Massey

Wikipedia: *James Lee Massey* (1934 – 2013) was an information theorist and cryptographer, Professor Emeritus of Digital Technology at ETH Zurich. His notable

work includes the application of the Berlekamp–Massey algorithm to linear codes, the design of the block ciphers IDEA (with Xuejia Lai) and SAFER, and the Massey-Omura cryptosystem (with Jim K. Omura).

Elwyn Berlekamp

Wikipedia: *Elwyn Ralph Berlekamp* (1940) is an American mathematician. He is a professor emeritus of Mathematics and EECS at the University of California, Berkeley. Berlekamp is known for his work in coding theory and combinatorial game theory.
He is a member of the National Academy of Engineering (1977) and the National Academy of Sciences (1999). He was elected a Fellow of the American Academy of Arts and Sciences in 1996, and became a fellow of the American Mathematical Society in 2012. In 1991, he received the IEEE Richard W. Hamming Medal, and in 1993, the Claude E. Shannon Award. In 1998, he received a Golden Jubilee Award for Technological Innovation from the IEEE Information Theory Society.

Let us demonstrate the Berlekmap-Massey algorithm on a small example:

Consider the sequence

$\{s_i\}_{i=0}^{30} = \{0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0\}$.

All intermediate functions will also be printed below.

```
s = {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
   1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0};
Lol = 0; fol = 1;
diff = 0; Clear[x];
f = 1; L = 0; g = CoefficientList[f, {x}];
Do[If[Mod[Sum[g[[i]] s[[j - 1 - L + i]], {i, 1, L}], 2] == s[[j]], diff = diff + 1,
        Lne = Max[j - L, L];
        fne = PolynomialMod[x^(Lne-L) f + x^(Lne-Lol-diff-1) fol, 2];
  If[Lne ≠ L, fol = f; Lol = L; L = Lne; diff = 0,
   diff = diff + 1]; f = fne; g = CoefficientList[f, {x}]];
 Print["j=", j, ", L=", L, ", f=", f], {j, Length[s]}]
```

j=1, L=0, f=1

j=2, L=0, f=1

j=3, L=0, f=1

j=4, L=0, f=1

j=5, L=0, f=1

j=6, L=6, f=$1 + x^6$

j=7, L=6, f=$1 + x^5 + x^6$

j=8, L=6, f=$1 + x^5 + x^6$

j=9, L=6, f=$1 + x^5 + x^6$

j=10, L=6, f=$1 + x^5 + x^6$

j=11, L=6, f=$1 + x^5 + x^6$

j=12, L=6, f=$x^5 + x^6$

j=13, L=6, f=$x^5 + x^6$

j=14, L=6, f=$x^5 + x^6$

j=15, L=6, f=$x^5 + x^6$

j=16, L=6, f=$x^5 + x^6$

j=17, L=6, f=$x^5 + x^6$

$j=18,\ L=12,\ f=1+x^{11}+x^{12}$

$j=19,\ L=12,\ f=1+x^{10}+x^{12}$

$j=20,\ L=12,\ f=1+x^{9}+x^{12}$

$j=21,\ L=12,\ f=1+x^{8}+x^{12}$

$j=22,\ L=12,\ f=1+x^{7}+x^{12}$

$j=23,\ L=12,\ f=1+x^{6}+x^{12}$

$j=24,\ L=12,\ f=1+x^{5}+x^{12}$

$j=25,\ L=13,\ f=x+x^{5}+x^{13}$

$j=26,\ L=13,\ f=1+x+x^{12}+x^{13}$

$j=27,\ L=14,\ f=1+x+x^{2}+x^{5}+x^{12}+x^{13}+x^{14}$

$j=28,\ L=14,\ f=x^{2}+x^{5}+x^{14}$

$j=29,\ L=14,\ f=x^{2}+x^{5}+x^{14}$

$j=30,\ L=16,\ f=1+x+x^{4}+x^{7}+x^{12}+x^{13}+x^{16}$

$j=31,\ L=16,\ f=1+x+x^{4}+x^{7}+x^{12}+x^{13}+x^{16}$

5 Authentication codes

See Latex document.

6 Error-Correcting Codes

6.6.1 Binary cyclic codes

6.6.2 Decoding More Errors Efficiently

We will now look into the decoding of the binary BCH code C of length $n = 2^m - 1$ with designed distance $d = 2t + 1$.

As before let α be a primitive element in $\mathbb{F}_{2^m} = \mathrm{GF}(2^m)$.

The defining relation of the code C is:

$$c(x) \in C \quad \Leftrightarrow \quad c(\alpha) = c\left(\alpha^2\right) = \ldots = c\left(\alpha^{2t}\right) = 0. \tag{6.1}$$

Suppose that $c(x)$ is a transmitted codeword and that $r(x)$ is received.

The error vector follows from $r(x) = c(x) + e(x)$. Let $e(x)$ have weight u with $u \le t$, by assumption.

The locations of the errors are denoted by the *field error locations* X_j, $1 \le j \le u$.

In other words, if the locations of the errors are given by $i_1 < i_2 < \ldots \le i_u$, i.e. if $e(x) = x^{i_1} + x^{i_2} + \cdots + x^{i_u}$, then the field error locations are given by

$$X_1 = \alpha^{i_1}, \qquad X_2 = \alpha^{i_2}, \qquad \ldots \qquad X_u = \alpha^{i_u}.$$

Let

$$S_j = r\left(\alpha^j\right), \qquad j \ge 0. \tag{6.2}$$

The syndrome of the received word $r(x)$ is given by $S_1, S_2, \ldots, S_{2t}$ and satisfies by (6.2)

$$S_j = \quad r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j) =$$

$$\alpha^{j\,i_1} + \alpha^{j\,i_2} + \cdots + \alpha^{j\,i_u} = \sum_{i=1}^{u} X_i{}^j, \qquad 1 \le j \le 2\,t. \tag{6.3}$$

The goal is to find the X_i from these $2\,t$ equations with u minimal.

> **Definition 6.1**
> **The error locator polynomial $\sigma(z) = \sum_{i=0}^{u} \sigma_i\, z^i$ of the error vector is given by**
>
> $$\sigma(z) = \prod_{i=1}^{u} (1 - X_i\, z)$$

Once the error-locator polynomial is known, its zeroes tell where the errors are.

For instance, if $\sigma(z) = (1 - \alpha^5\, z)\,(1 - \alpha^{11}\, z)$, we know that two errors occured, namely at coordinates 5 and 11.

Consider the generating function

$$S(z) = \sum_{j=1}^{\infty} S_j\, z^j = \sum_{j=1}^{\infty} \sum_{i=1}^{u} X_i{}^j\, z^j = \sum_{i=1}^{u} \sum_{j=1}^{\infty} X_i{}^j\, z^j = \sum_{i=1}^{u} \frac{X_i\, z}{1 - X_i\, z}.$$

Multiply left and right hand sides with $\sigma(z)$:

$$S(z)\,\sigma(z) = \sum_{i=1}^{u} \frac{X_i\, z}{1 - X_i\, z} \prod_{j=1}^{u} (1 - X_j\, z) = \sum_{i=1}^{u} X_i\, z \prod_{j \ne i} (1 - X_j\, z).$$

Add $\sigma(z)$ to both sides to get

$$\sigma(z)\,(1 + S(z)) = \sigma(z) + \sum_{i=1}^{u} X_i\, z \prod_{j \ne i} (1 - X_j\, z).$$

The right hand side will be shortened to $\omega(z)$, a polynomial of degree at most u, the actual number of errors that occurred. So, by definition

$$\omega(z) = \sum_{j=0}^{e} \omega_j\, z^j = \sigma(z) + \sum_{i=1}^{u} X_i\, z \prod_{j \ne i} (1 - X_j\, z).$$

and we have the relation

$$\sigma(z)\,(1 + S(z)) = \omega(z).$$

In view of (6.4), only S_1, S_2, …, $S_{2\,t}$ are known.

Theorem 6.1
Consider the *t*-error correcting BCH code of length *n* with *designed distance* $2t+1$ and with parity check matrix as given in (6, 1).

Let $r(x)$ be a received vector and S_j the syndrome defined by $S_j = r(\alpha^j)$, $1 \le j \le 2t$.

Put $S(z) = \sum_{j=1}^{\infty} S_j z^j$.

Suppose that $r(x)$ is not a codeword and that $u \le t$ errors have been made, say at locations X_j, $1 \le j \le u$.

Let the error locator polynomial $\sigma(z)$ be defined by Def. 6.2. Then

$$\sigma(z)\,(1 + S(z)) \equiv \omega(z) \pmod{z^{2t+1}},$$

where $\sigma(z)$ and $\omega(z)$ are polynomials of degree u, resp at most u, and where $\sigma_0 = 1$.

This equation is called the key equation. Finding $\sigma(z)$ and $\omega(z)$ from this equation amounts to decoding $r(x)$.

There are many algorithms known in the literature to solve the key equation efficiently. One of these algorithms makes use of the extended version of Euclid's Algorithm applied to the polynomials $1 + S(z)$ and z^{2t+1}.

Berlekamp (1967) does it iteratively. He solves

$$\sigma^{(i)}(z)\,(1 + S(z)) \equiv \omega^{(i)}(z) \pmod{z^{i+1}}$$

for $i = 0, 1, \ldots, 2t$.

Massey (1969) saw the similarity of the key equation with equation $S(z)\,f^*(z) = \omega(z)$ for the output sequence $S_0, S_1, S_2 \ldots$ of linear feedback shift registers with unknown characteristic polynomial $f(z)$ of degree n (the degree of $\omega(z)$, also unknown, is less than n).

He showed that Berlekamp's iterative decoding algorithm can be translated to a technique to determine the linear complexity of a binary sequence.

It is called the Berlekamp-Massey algorithm.