Generic Analysis of Small Cryptographic Leaks

Adi Shamir
Computer Science Dept
The Weizmann Institute
Israel
(Joint work with Itai Dinur)

Side Channel Attacks

- Are extremely powerful, and in many cases are the only practical way to break well designed cryptosystems

- Had been studied for more than a decade in academia, and for much longer by others

- Many types of side channel attacks are known, but each one needs different physical and mathematical techniques

- Still lacks a satisfactory unifying framework

The typical Scenario Considered So Far:

- A new type of potential leakage is discovered, which provides a very small amount of very indirect information about the cryptographic key
- Specialized techniques have to be developed to extract the full key from a large number of measurements of this new source of information
- To apply it to a particular device, detailed information about the physical and logical implementation of the cryptosystem in that device is usually required
- The success of each attack is extremely sensitive to the existence of unknown countermeasures

My Goal in This Talk:

- To develop a generic way how to analyze any new type of side channel leakage
- Applying the attack will not require detailed knowledge of the physical and logical implementation of the cryptosystem
- However, its success will not be guaranteed, and will have to be tested experimentally in each case

Examples of Possible scenarios:

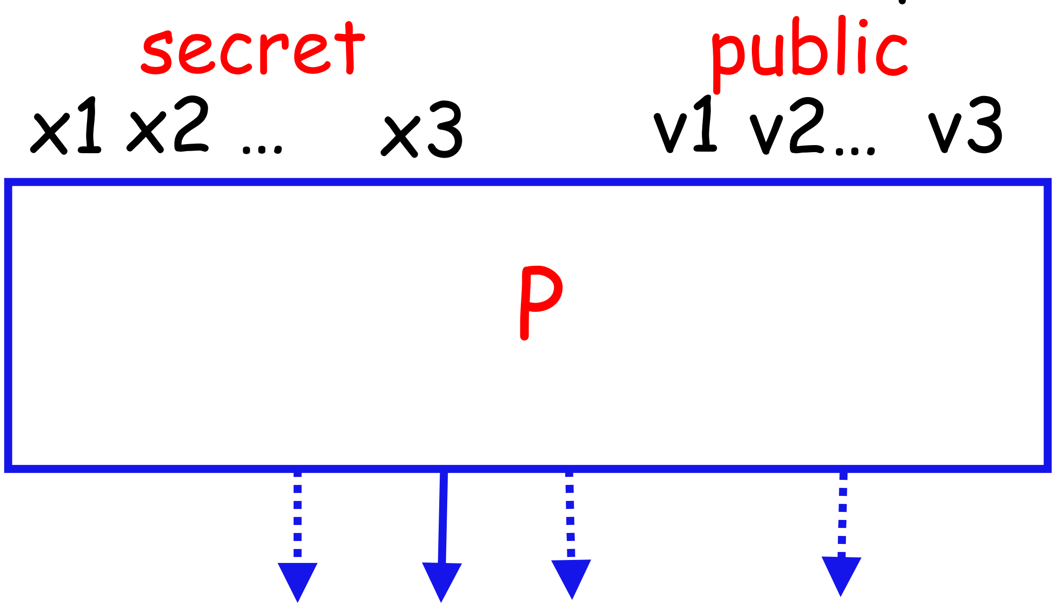
- We are given a chip, and can probe any wire in it. However, we have no idea what kind of data is passing through the wire during each cycle
- We can measure the total power consumption of the chip, but do not know how this power consumption is related to the instructions executed by the processor or to the data operated upon
- We can use a tiny antenna to measure the RF field near the surface of the chip, but do not know how this field is related to the crypto key

The new CUBE ATTACK (Dinur&Shamir):

- Is a very general key derivation algebraic attack
- Generalizes and improves some previous summation-based attacks such as Integral Attacks and Vielhaber's AIDA
- Was recently used to break the full version of the Grain-128 stream cipher
- As we show in this talk, cube attacks are ideal generic tools which can be applied in principle to any type of side channel leakage

Any cryptographic scheme can be described by multivariate polynomials:

- Each computed bit can be described by some multivariate polynomial $P(x_1,\ldots x_n,v_1,\ldots v_m)$ over GF(2) of secret variables x_i (key bits), and public variables v_j (plaintext bits in block ciphers/MAC's, IV bits in stream ciphers)



The main characteristics of cryptographically defined polynomials:

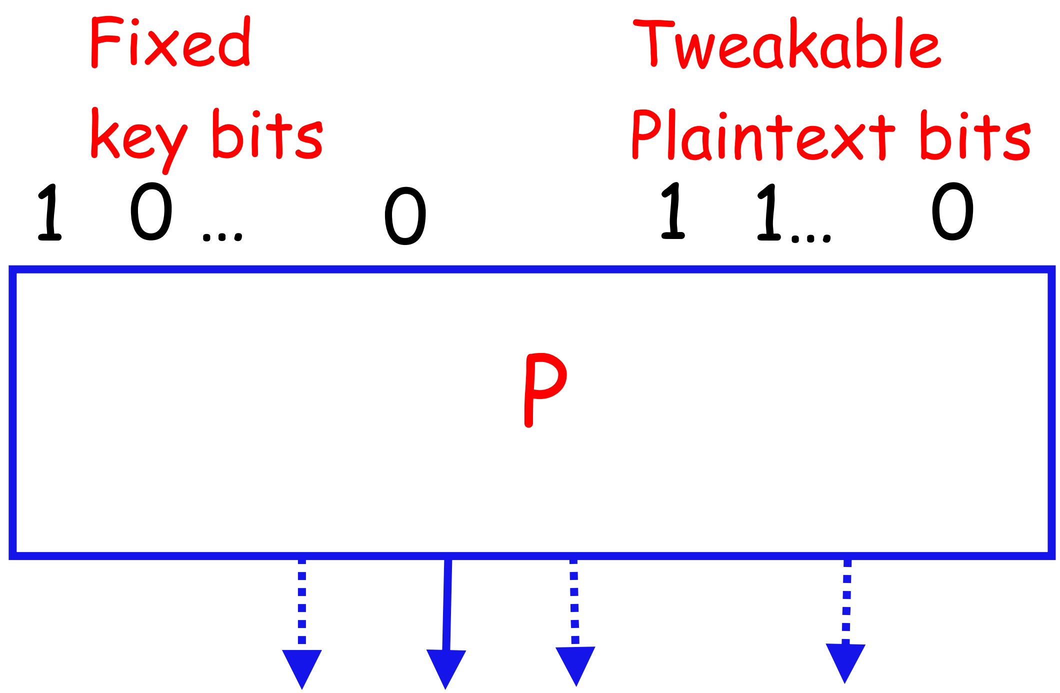
- We consider only multivariate polynomials in fully expanded Algebraic Normal Form

- These polynomials are typically huge, and can not be explicitly defined, stored, or manipulated with a feasible complexity

- The data available to the attacker will typically be insufficient to interpolate their coefficients from their output values

Black box multivariate polynomials:

The only realistic way to deal with these polynomials is as black box polynomials, which can be evaluated on any (fully specified) set of secret and public inputs:



The typical problem of algebraic cryptanalysis:

- Solve a system of black box polynomial equations over GF(2):

  $P_1(x_1 \ldots x_n v^1{}_1 \ldots v^1{}_m)=0$

  $P_2(x_1 \ldots x_n v^2{}_1 \ldots v^2{}_m)=1$

  $P_3(x_1 \ldots x_n v^3{}_1 \ldots v^3{}_m)=0$

  …

  in which the fixed key variables x_i are unknown, and the various plaintext/IV variables $v^j{}_i$ are known
- The problem is NP-hard and exceedingly difficult in practice, even with explicitly given polynomials

The new cube attack:

- Can be applied directly to arbitrary black box polynomials, even when they are huge
- Can be applied to unknown or partially known cryptographic schemes given as black boxes
- Can be applied automatically without careful preanalysis of the properties of the scheme
- Is provably successful when the black box polynomials are sufficiently random

Cube attacks have two phases:

- A preprocessing phase (via simulation):
  - The cryptosystem is given as a black box. The attacker can obtain one bit of output for any chosen key and plaintext.
- The online phase (via eavesdropping):
  - The cryptosystem is given as a black box, with the key set to a secret fixed value. The attacker can obtain one bit of output for any chosen plaintext.

The complexity of the attack:

- For random polynomials of degree d in n input variables over GF(2), the complexity of cube attacks is $O(n2^{d-1}+n^2)$ bit operations, which is polynomial in the key size n (!)

- Bits of information leaking out during the early stages of the encryption process are likely to be described by low degree polynomial functions in the plaintext and key bits, making the attack feasible

A typical example of a cube attack:

- To demonstrate the attack, consider the following dense master polynomial of degree $d=3$ over three secret variables x_1,x_2,x_3 and three public variables v_1,v_2,v_3:

$P(v_1,v_2,v_3,x_1,x_2,x_3)=$

$$v_1v_2v_3+v_1v_2x_1+v_1v_3x_1+v_2v_3x_1+v_1v_2x_3+v_1v_3x_2+ v_2v_3x_2+v_1v_3x_3+v_1x_1x_3+v_3x_2x_3+x_1x_2x_3+v_1v_2+ v_1x_3+v_3x_1+x_1x_2+x_2x_3+x_2+v_1+v_3+1$$

The effect of partial substitution:

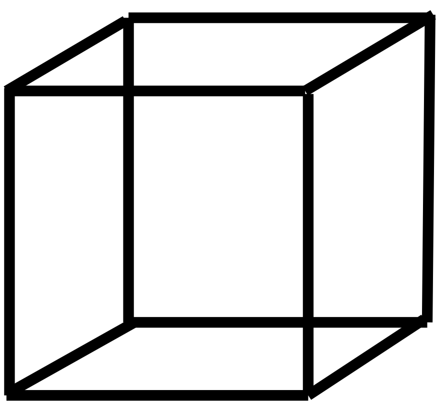
- Substituting $v_1=1$ and $v_2=1$, we get a derived symbolic polynomial in the remaining variables x_1,x_2,x_3 and v_3:

$$P(v_1,v_2,v_3,x_1,x_2,x_3)=$$

$$x_1+x_2+v_3x_1+v_3x_3+x_1x_2+x_2x_3+x_1x_3+v_3x_2x_3 +x_1x_2x_3+ 1$$

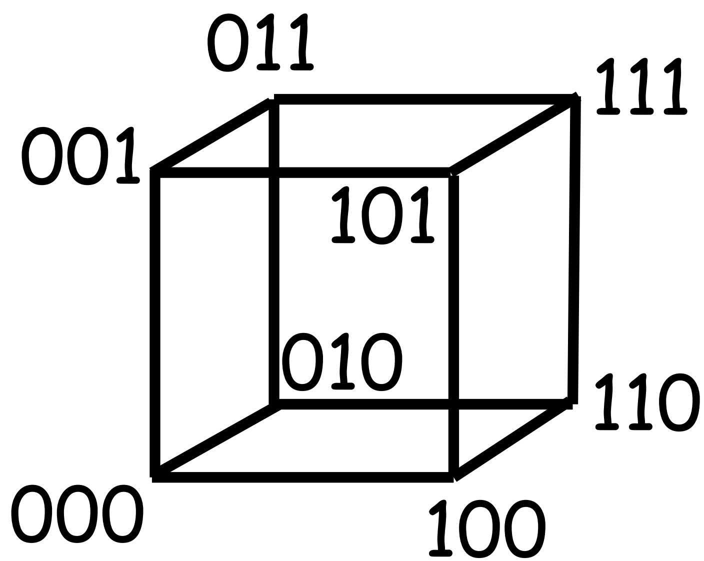
The Boolean cube:

Each corner of the Boolean cube will have 3 interpretations in cube attacks:



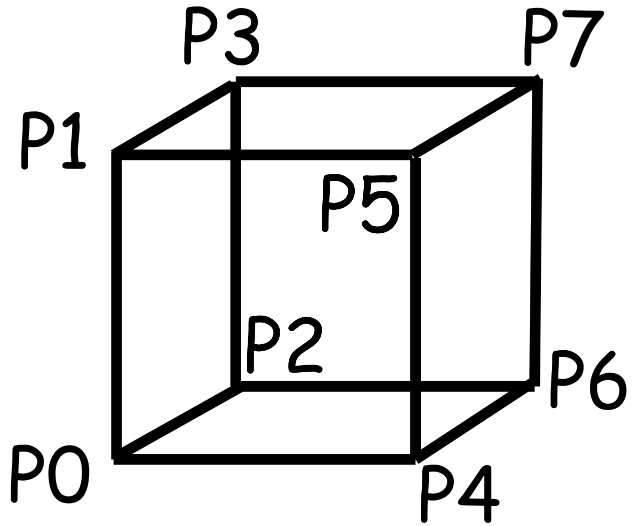
The Boolean cube:

An assignment of 0/1 values to some subset of the public v_j variables



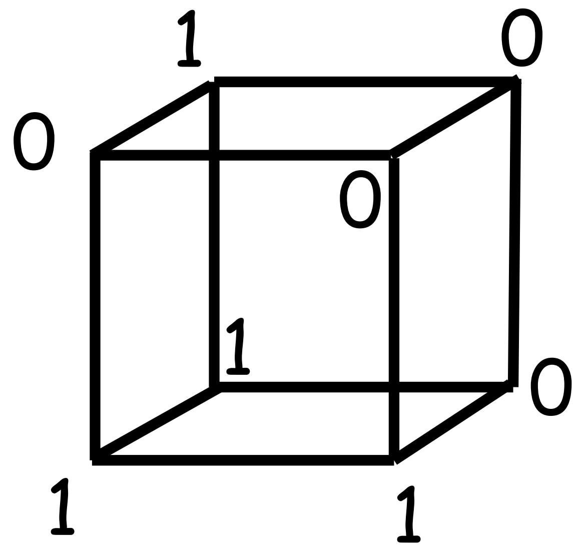
The Boolean cube:

The simplified symbolic form of the corresponding derived polynomial



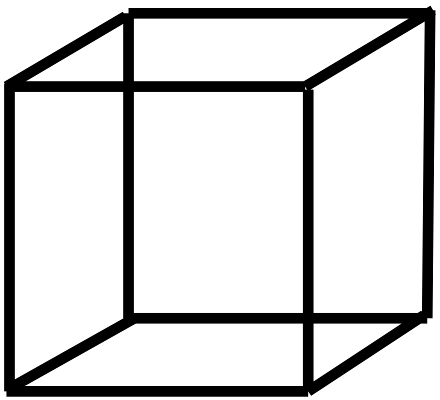
The Boolean cube:

The 0/1 value of this derived polynomial when all the other variables are set to their public and secret values

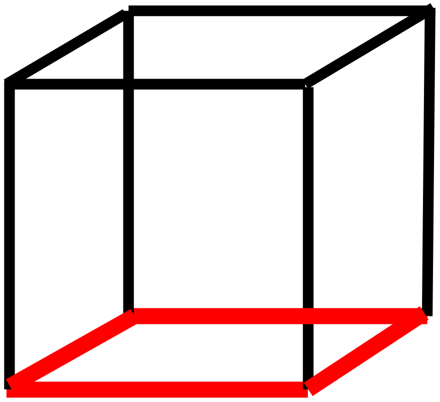


The Boolean cube:

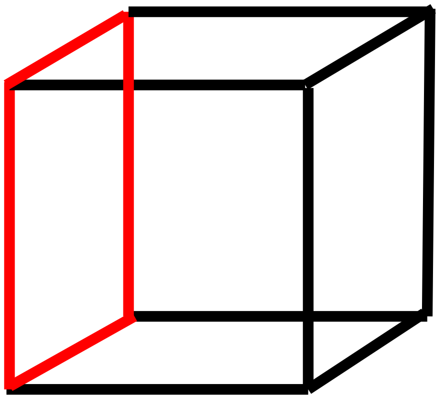
We sum over GF(2) both the symbolic forms of the derived polynomials and their 0/1 values which occur in the vertices of various (potentially overlapping) subcubes



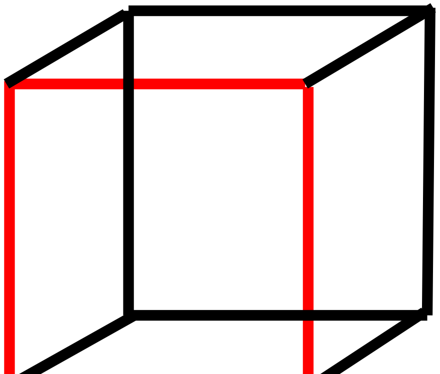
The summations:



The summations:



The summations:



In our small example:

- Summing the 4 derived polynomials with $v_1=0$, all the nonlinear terms disappear and we get x_1+x_2; summing the 4 derived polynomials with $v_2=0$ we get $x_1+x_2+x_3$; and summing the four derived polynomials with $v_3=0$ we get x_1+x_3
- The sums of polynomials equated to their summed values give rise to three linear equations in the three secret variables x_i, which can be easily solved

Why did all the nonlinear products of secret variables disappear from the sum?

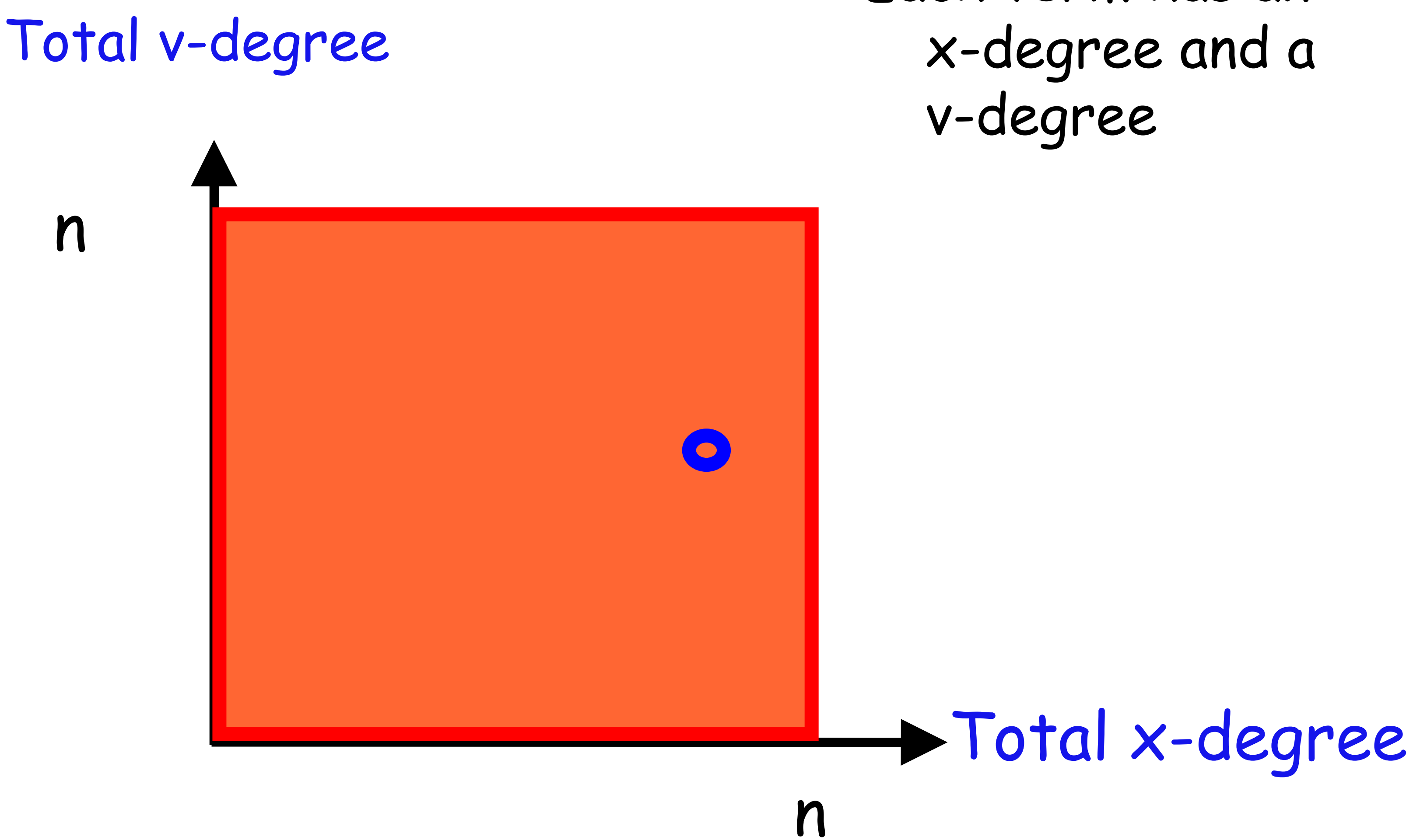
- All the terms are the products of at most 3 of the 6 x_i and v_j variables
- We sum over all the values of two v_j's
- Any term in the master polynomial P such as $x_1x_2v_1$ which contains the nonlinear product of two or more x_i in it, is missing at least one of the v_j that we sum over, and is thus added an even number of times modulo 2 to the sum

Isn't cube attack just a differentiation? No wonder that it reduces the degree…

- However, each terms has two types of variables: $v_1v_2v_4x_2x_3x_4$
- What we want: to reduce the x-degree to linear
- What we can do: to reduce the v-degree by differentiation
- Differentiating the term above wrt v_1v_2 gives $v_4x_2x_3x_4$; wrt v_1v_3 gives 0; neither has x-degree 1.

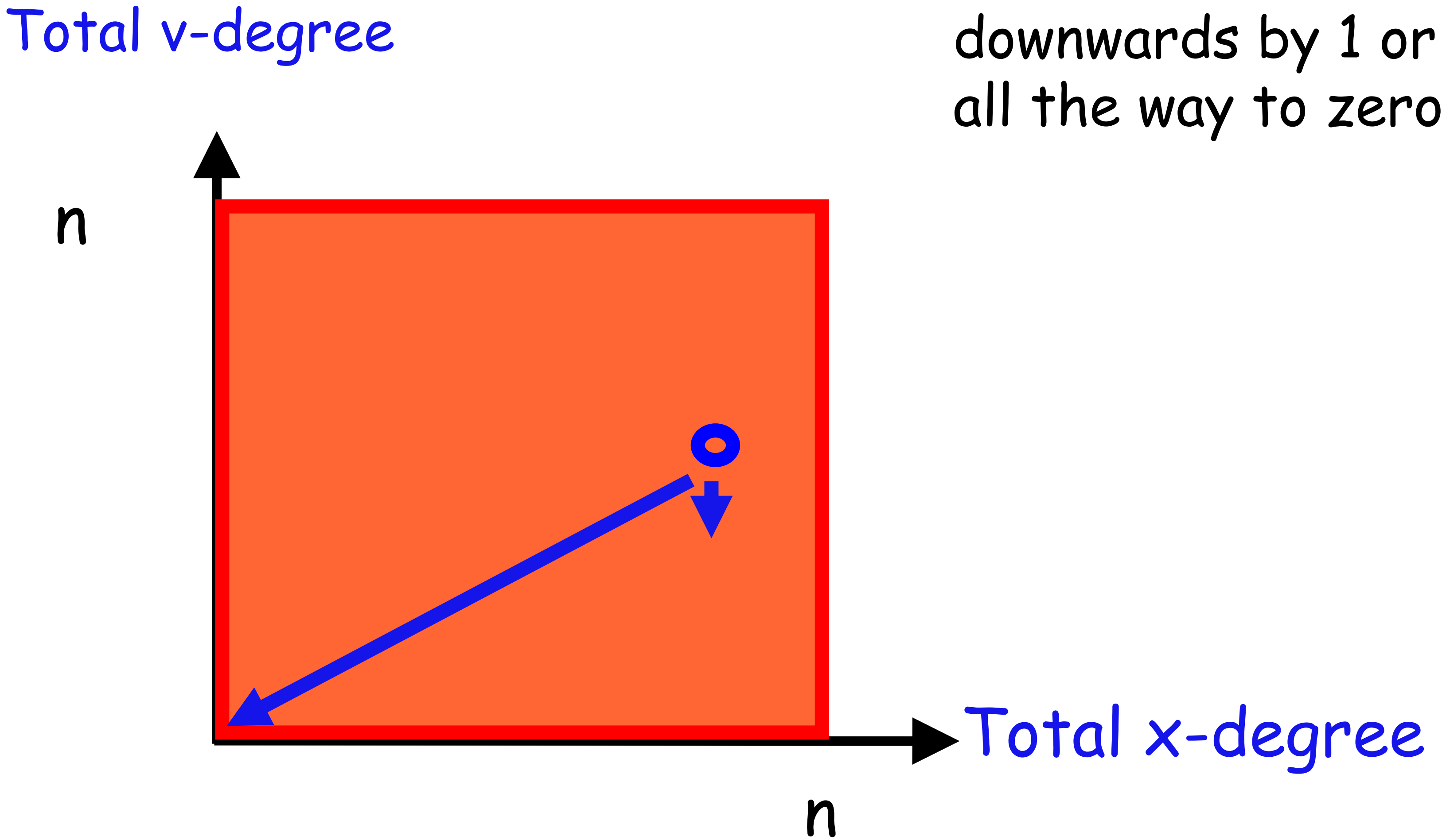
Consider a general polynomial in n secret and n public variables:

Each term has an x-degree and a v-degree



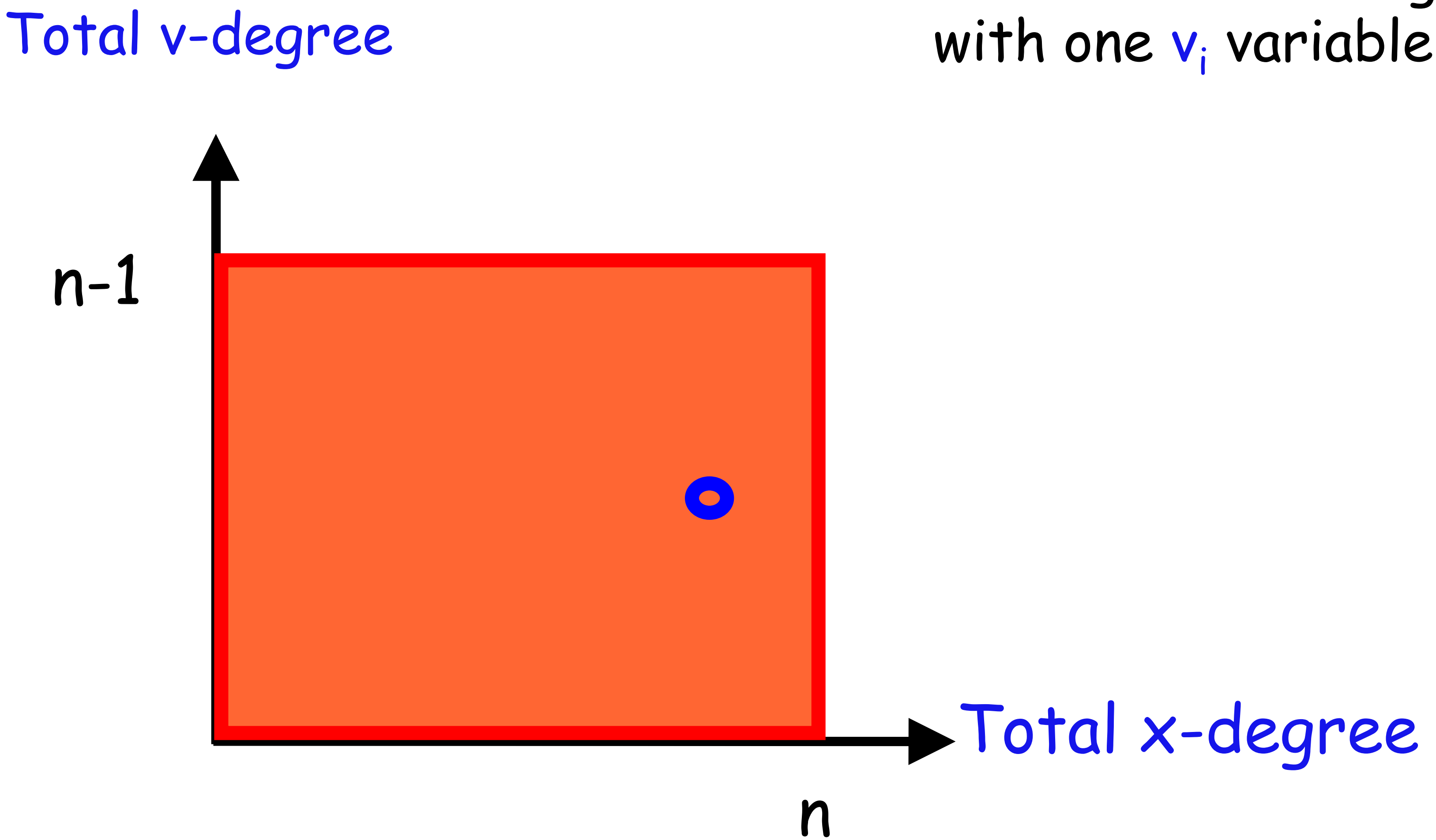
Differentiating wrt public variables reduce v-degrees

Each term moves downwards by 1 or all the way to zero



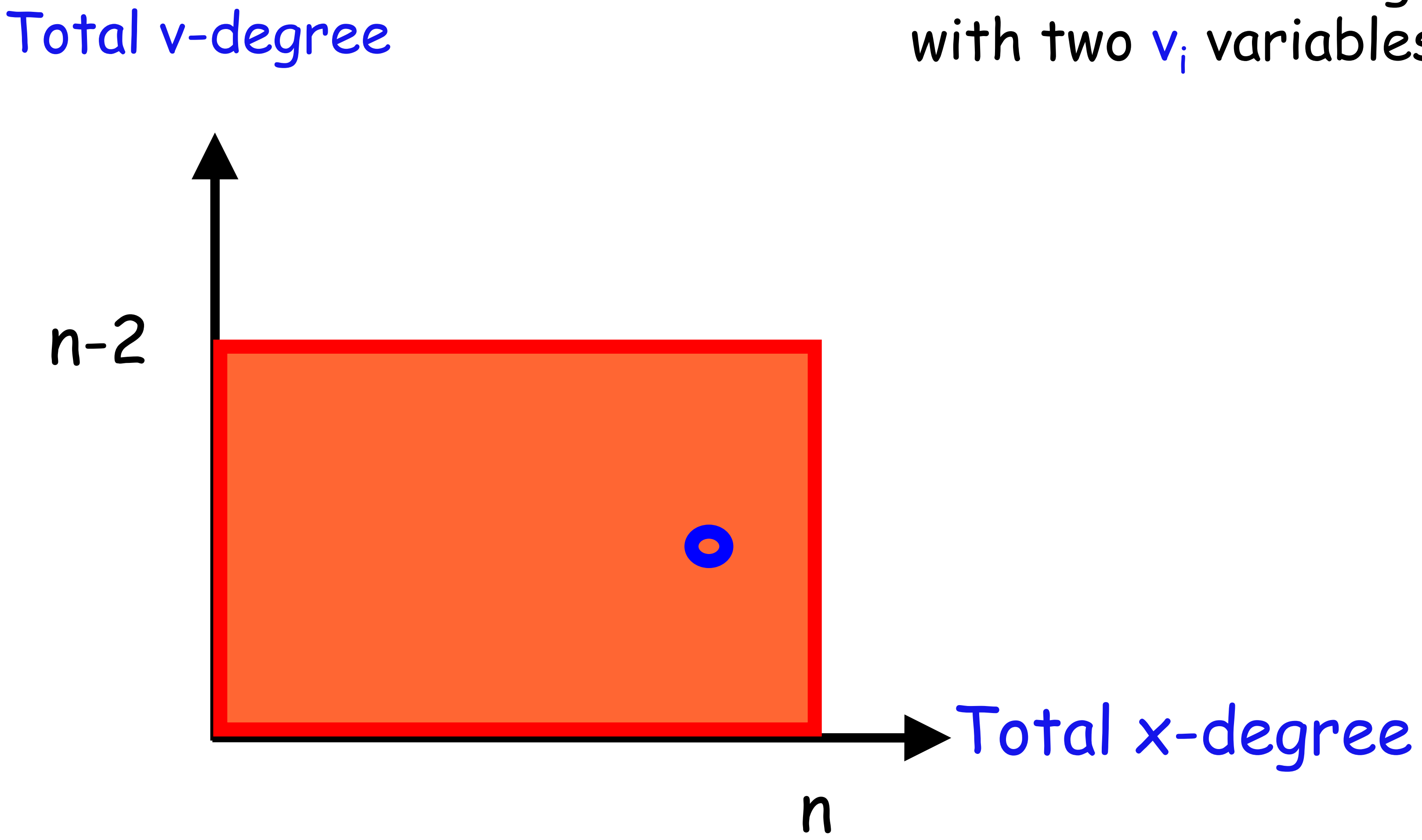
Differentiating wrt public variables reduce v-degrees

After differentiating with one v_i variable



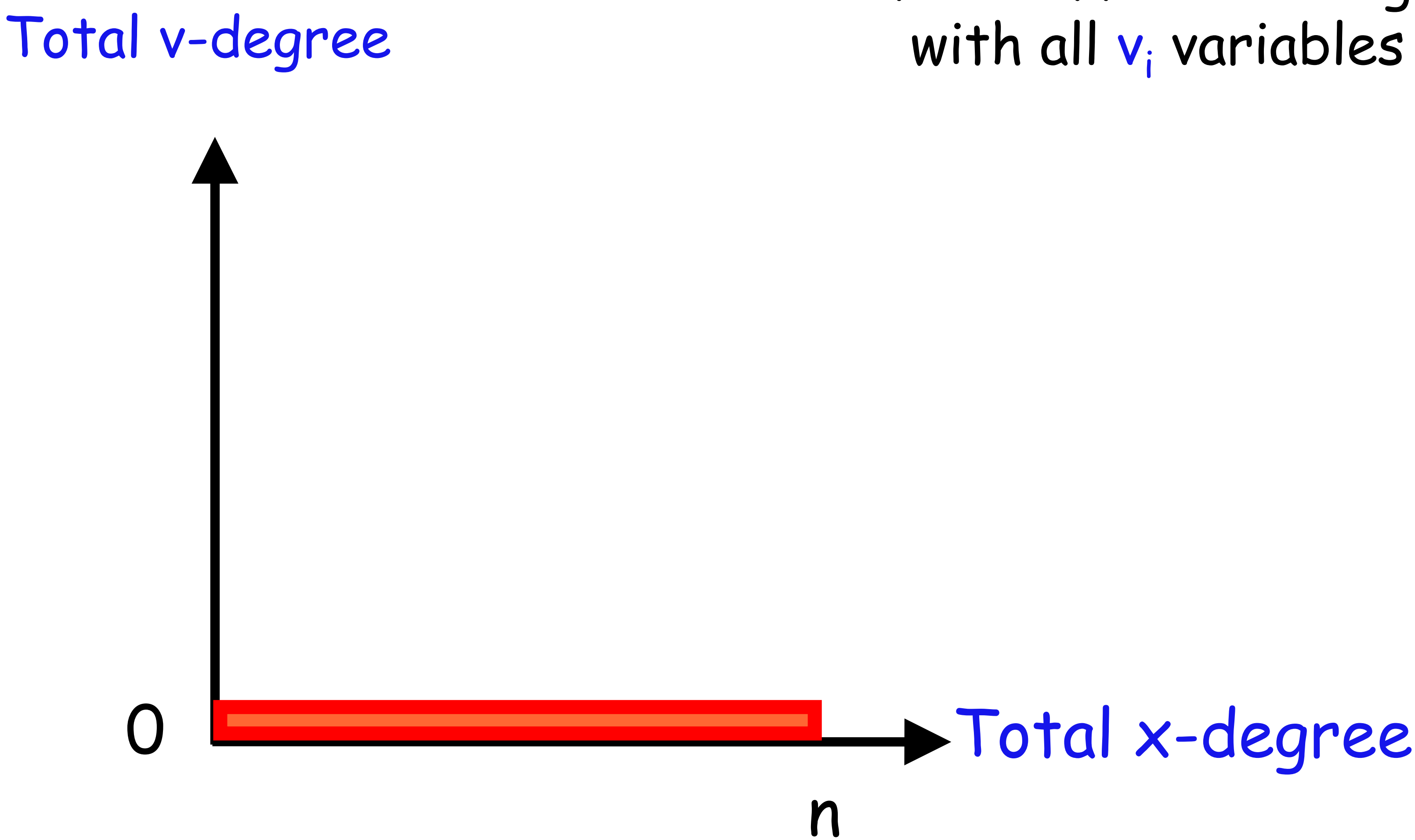
Differentiating wrt public variables reduce v-degrees

After differentiating with two v_i variables



A general polynomial will still have x-degree of n even after differentiating wrt all its public variables

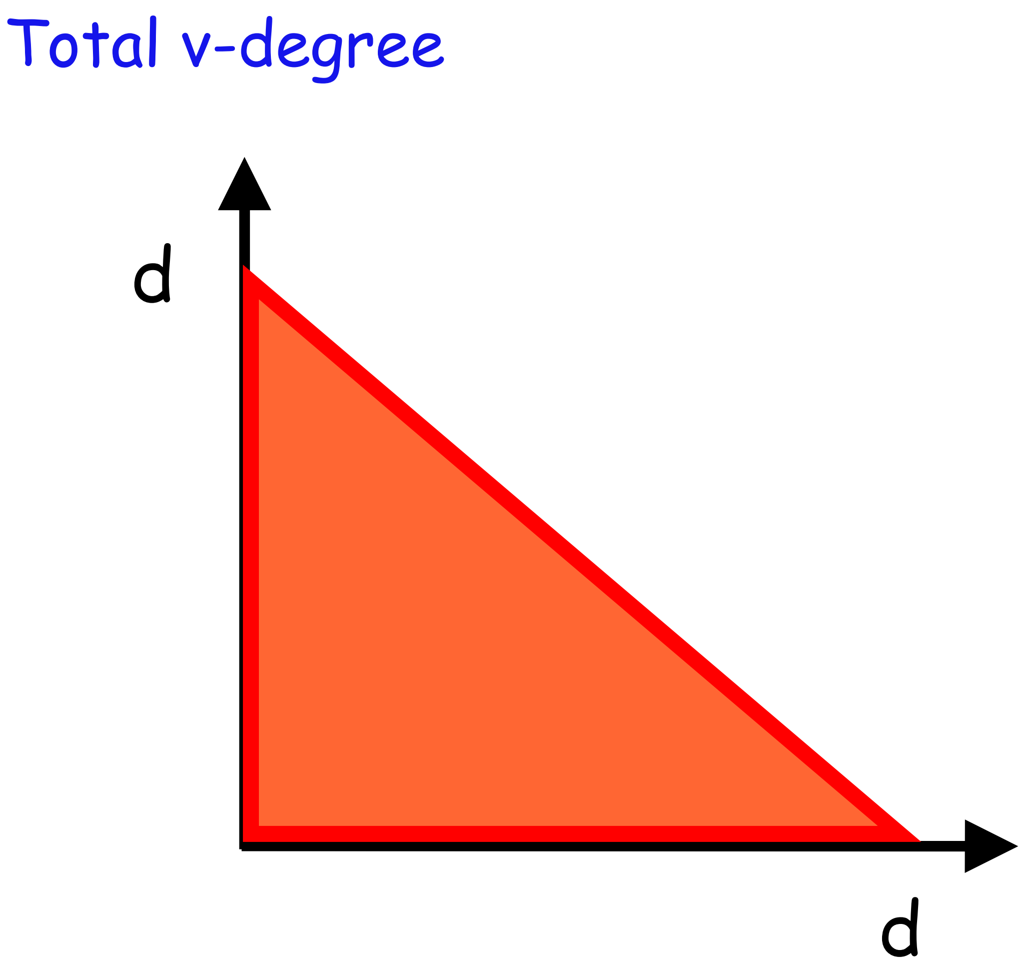
After differentiating with all v_i variables



In cube attacks, we consider general polynomials of total degree $d<n$ in all the public and secret variables

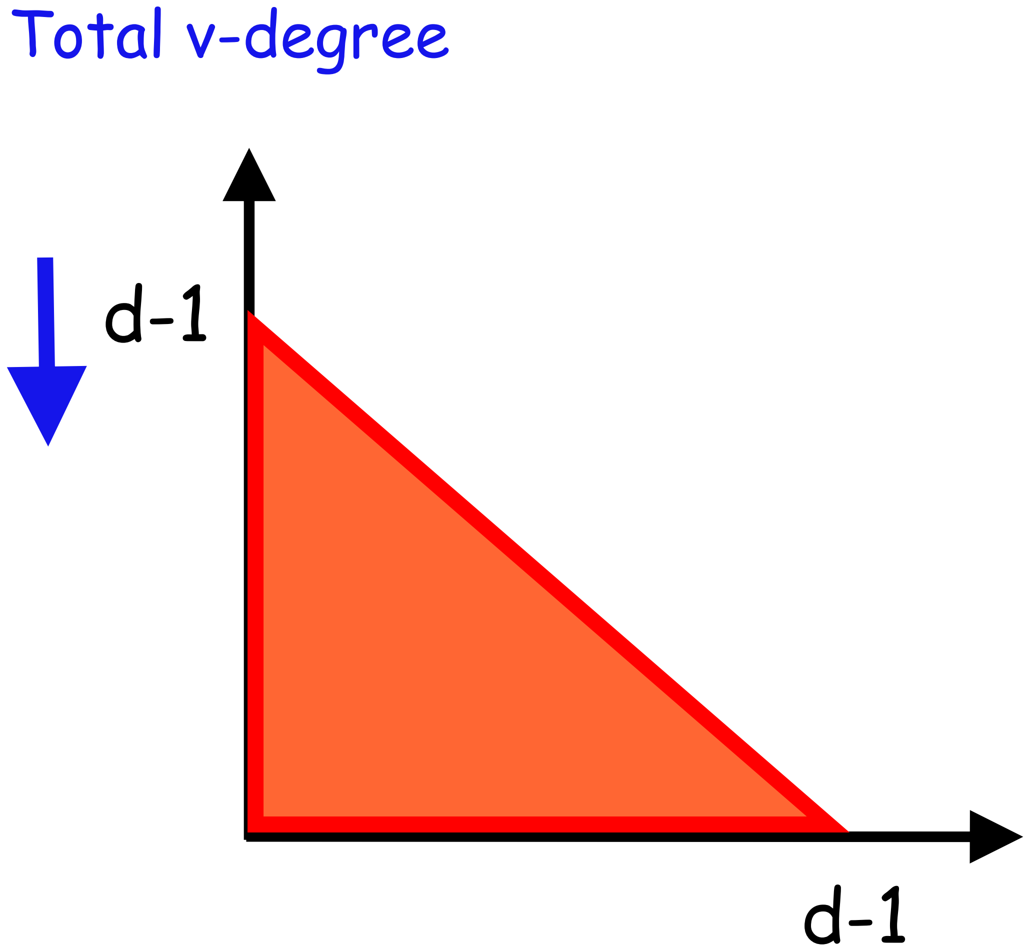
In cube attacks, we consider general polynomials of total degree d<n in all the public and secret variables

Our polynomials have triangular shape:



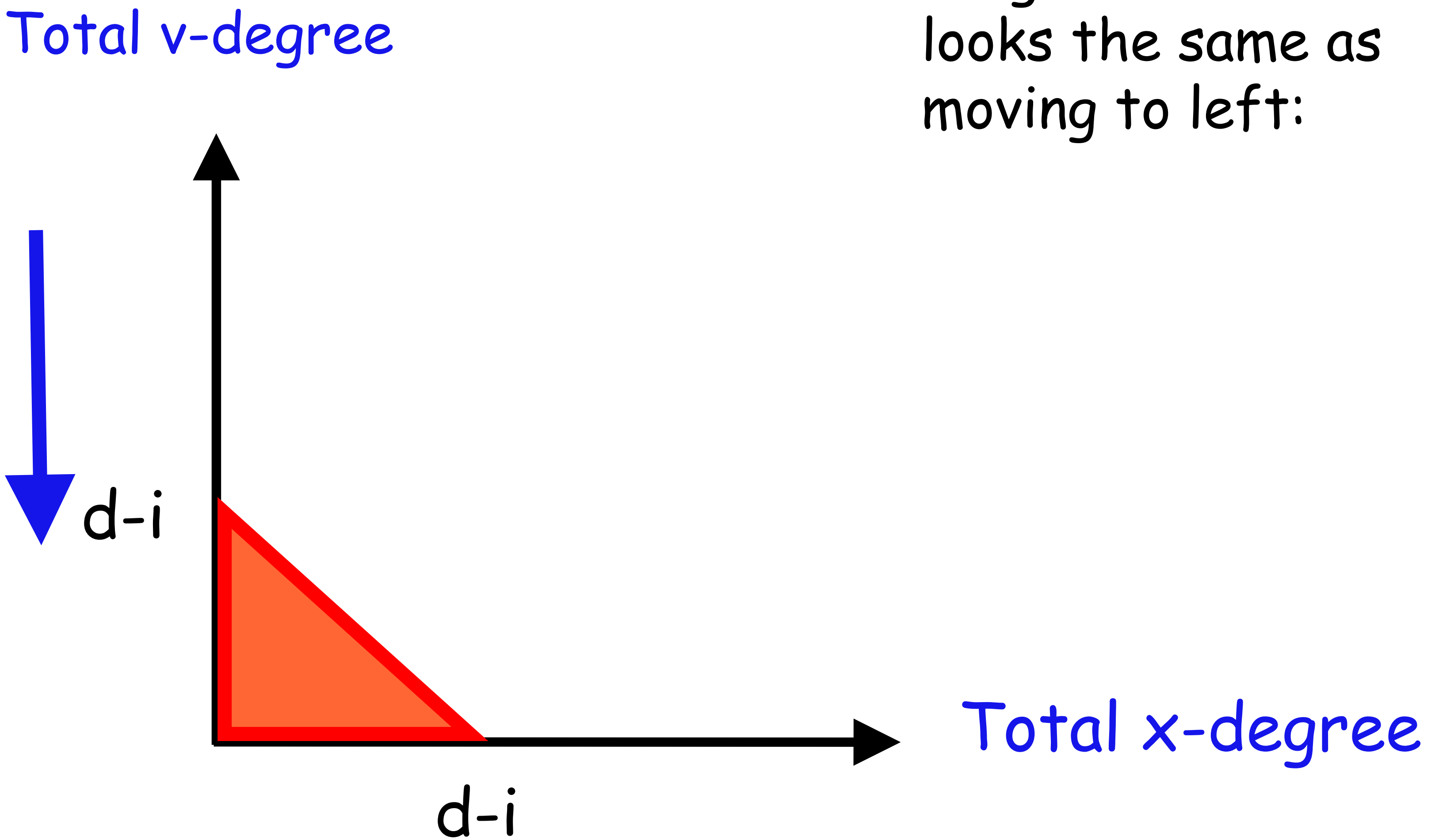
Differentiating with respect to one public variable:

Moving downwards looks the same as moving to left:



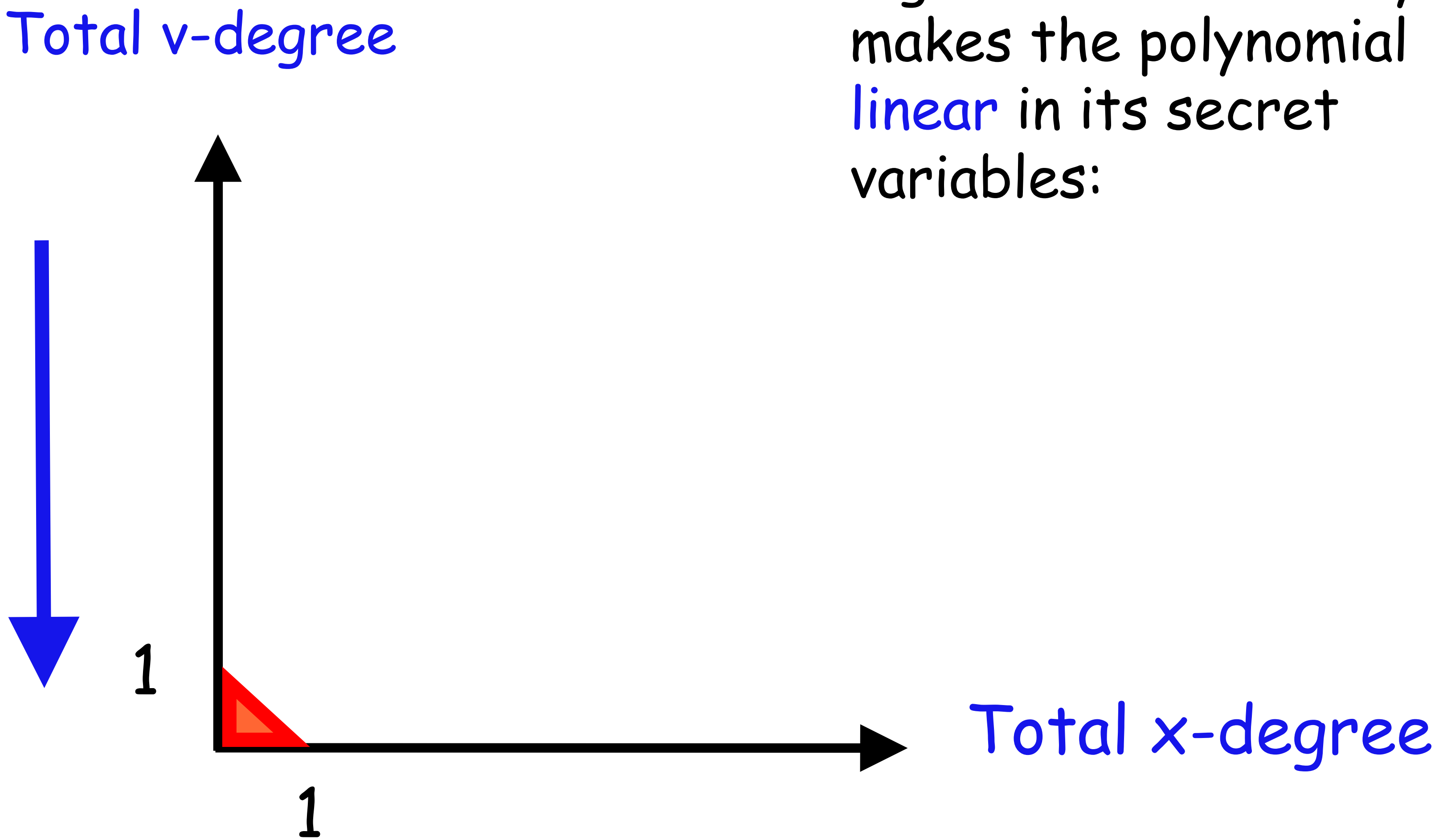
Differentiating with respect to i public variables:

Moving downwards looks the same as moving to left:



Differentiating with respect to d-1 public variables:

Going almost all the way makes the polynomial linear in its secret variables:



How to find the indices to sum over:

- The derived polynomials cannot be explicitly generated or symbolically summed from the master polynomial with feasible complexity
- We use the preprocessing phase (which is executed only once for each cryptosystem) to experimentally find the best choice of summation indices. Note that during preprocessing, the attacker is allowed to choose both the key and IV variables

Robust Cube Attacks:

- Cube attacks typically XOR millions of bits in order to compute the right hand side of each linear equation
- This is ok when the bits are high quality bits obtained from actual ciphertexts
- This is problematic when the bits have even 0.0001% noise, and thus even SMALL AMOUNT OF NOISE is a BIG PROBLEM in side channel attacks

Robust Cube Attacks:

- Fortunately, the attacker often knows which side channel information bits are potentially problematic
- The measured information is usually analog, whereas the information bits are digital
- Measurements which are near the quantization threshold are likely to contain most of the measurement errors

Robust Cube Attacks: The New Ingredients

- Cube attacks can usually provide an overdefined systems of linear equations by using a larger number of subcubes (random polynomials have exponentially many choices of summation indices, and we need only linearly many to solve for the key bits)
- Furthermore, the subcubes can overlap and reuse the same measured values, taking into account that they are the same value everywhere, even though they are unknown

Robust Cube Attacks: The New Ingredients

- The error correction problem is related to erasure codes, which provide information such as 011?10010?01101001110?100001

- Problem: By eliminating the problematic measurements, we lose the perfect cube structure, and thus the summation of the algebraic equations over just the good values will not result in linear equations!

Robust Cube Attacks:

- The robust attack can assign a new variable name z_i to each measurement which is known to be potentially unreliable

- The cube summation will have a right hand side which is the XOR of all the good bit values in the cube, plus the sum of all the variables which occurred in the subcube:

$$x_2+x_5+x_6+x_9=1+z_3+z_7+z_8$$

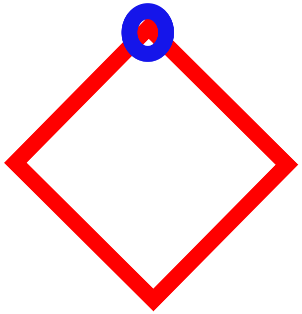
Robust Cube Attacks:

- The new trick: Use the numerous trivial equations of the form 0=0 obtained by summing over too many public variables
- $0 = 1+z_3+z_7+z_8$
- They are useless in order to find the key, but great for correcting all the errors.

Robust Cube Attacks: The New Ingredients

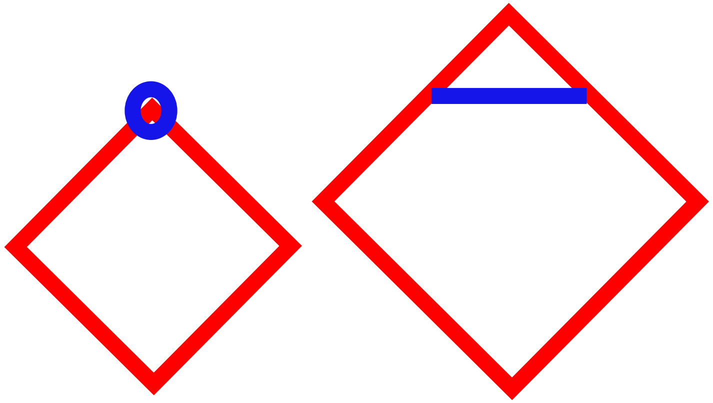
- In standard cube attacks, we get linear equations from the original degree d polynomial by summing over d-1 dimensional cubes, which differentiate the multivariate polynomial d-1 times

- Consider the collection of all subsets of d-1 vars:



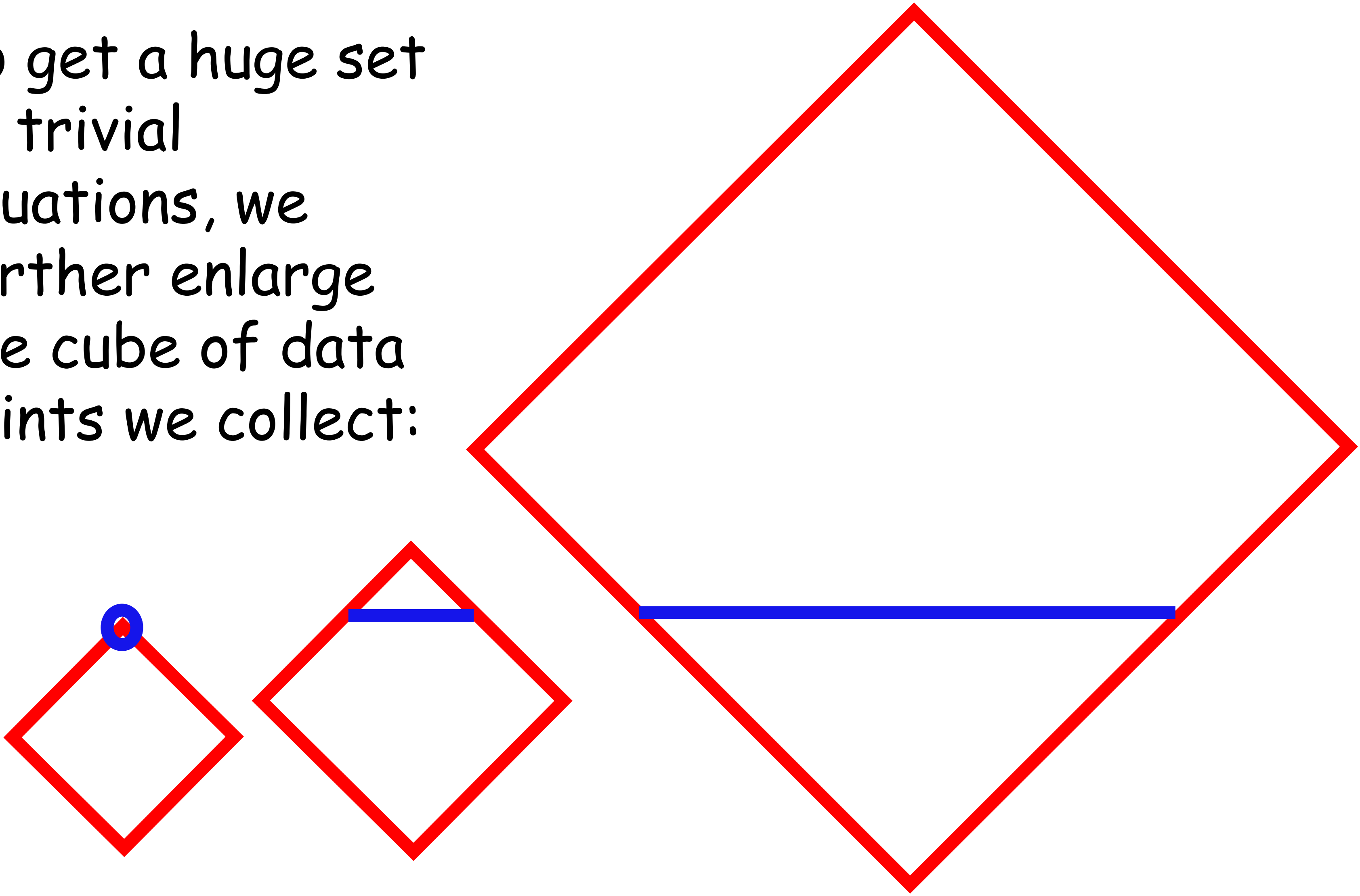
Robust Cube Attacks: The New Ingredients

- However, we want n rather than one linear equation, so we collect data from a slightly larger cube of about $n+\log(n)$ possible variables



Robust Cube Attacks: The New Ingredients

- To get a huge set of trivial equations, we further enlarge the cube of data points we collect:



Robust Cube Attacks: The New Ingredients

- We use a larger cube of dimension k. There are about 2^k-k^d subcubes of dimension >d within it

- Assuming that there is a fixed fraction e of known error locations in the large cube, there is a total of about $e2^k$ new variables that we have to add

- Simple computation shows that for random polynomials we can tolerate any $e < 1$ by making k sufficiently large

Leakage Attacks on Block Ciphers:

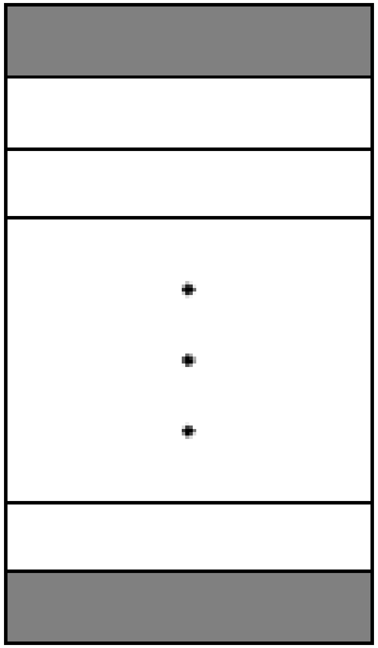
- Block ciphers are typically iterated, applying the same operations in each round to different values
- Any type of physical leakage is likely to repeat itself in each round, and all these values will be available to the cryptanalyst

Leakage Attacks on Block Ciphers:

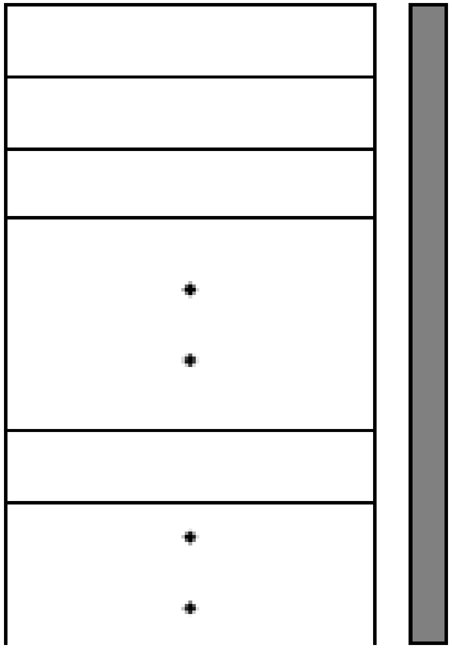
- The simplest type of leakage we consider is a single state bit, obtained e.g., by probing a single register cell or a single wire
- Another type of leakage is a single bit which is a simple function of many state bits, e.g., whether a carry occurred during an addition operation
- More complicated types of leakage can be multibit functions such as the Hamming weight of a byte written into memory

Information Available to the Attacker:

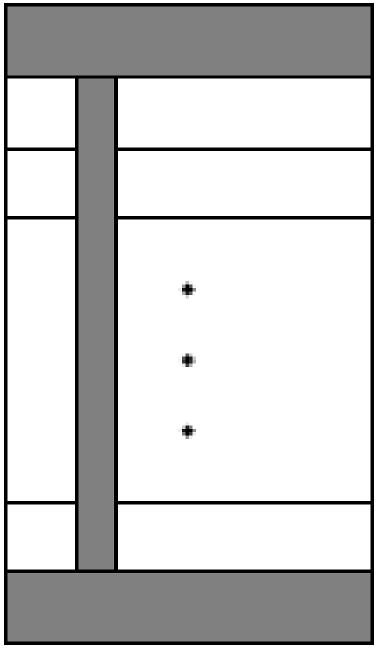
In block ciphers:



In stream ciphers:



In leakage attacks:



Which bits of information are useful?

- Single bits of information in successive rounds are difficult to relate to each other
- Our approach will be to relate a single bit of information to the fully known plaintext or ciphertext
- If the distance between them is too small, only few key bits can be typically extracted
- If the distance between them is too large, it is typically too difficult to get the key info

A Typical Example: AES-128

- A single bit of state data available after the initial whitening step $P+K_0$ reveals exactly one key bit
- A single bit of state data available after the first round is a function of one bit from K_1, together with at most 32 bits from K_0
- A single bit of state data after the second round depends on all the 128 key bits

A Typical Example: AES-128

- Our attack will only use the plaintext and a single state bit leaked from the end of the second round in multiple encryptions

- It will ignore the known ciphertext (which is too far from the state bit we analyze)

- It will ignore the state bits leaked during earlier/later rounds, since they add little information/are too difficult to analyze

A Typical Example: AES-128

- No previous type of attack (exhaustive/statistical/differential/linear) seems to be applicable in this scenario

- The new attack is completely practical, requiring about 2^{35} time for complete key recovery

- The mathematical part of the attack was simulated successfully on a single PC in a few minutes

Applying the cube leakage attack to AES:

- The preprocessing identified a collection of $n=128$ cubes with $d=28$ to sum over
- During the on-line attack on a particular key, we have to encrypt 2^7 sets of 2^{28} chosen plaintexts, and sum up the leaked bit in each set to determine the right hand side of each linear equation
- The total complexity of the attack is 2^{35}

Cube leakage attacks on SERPENT:

- Complete key avalanche in SERPENT occurs only at the end of the third round, due to the smaller 4-bit S-boxes and the weaker interaction between the state and key bits

- Since the degree of the polynomial grows more slowly in SERPENT than in AES, we were able to find n=128 cubes of dimension d=11

- The complexity of the attack is only $2^7 \times 2^{11} = 2^{18}$

Maxterms for 3-round Serpent:

Table 1. Maxterms for 3-round Serpent given the first state bit. Equations are given in the working key bits that are inserted to the first Sbox layer.

Maxterm Equation	Cube Indexes	Maxterm Equation	Cube Indexes
1+x0	{3,8,21,35,46,78,85,96,99,104,117}	x16+x48	{10,25,42,57,62,80,94,106,112,121,126}
x0+x96	{7,13,32,34,45,64,66,77,98,103,109}	1+x16+x112	{6,24,25,38,48,56,57,80,102,120,121}
x32	{7,13,34,45,64,66,77,96,98,103,109}	1+x48	{3,10,13,16,35,45,80,94,99,106,109}
x64	{0,2,8,34,36,40,68,96,98,100,104}	1+x80	{10,11,16,17,48,49,74,75,106,107,113}
x1+x33	{2,3,23,34,35,65,87,97,98,99,119}	x17+x49	{3,14,22,35,54,78,81,99,110,113,118}
x1+x97	{18,19,20,33,51,52,65,82,114,115,116}	x17+x113	{0,22,32,49,54,63,81,86,95,96,127}
1+x33	{1,18,19,20,50,51,52,65,82,115,116}	x49	{0,32,54,63,81,86,95,96,113,118,127}
1+x65	{1,18,19,20,33,50,51,52,82,115,116}	1+x81	{0,17,22,32,49,54,63,86,95,96,127}
1+x2	{5,16,37,48,58,76,90,98,101,112,122}	x18+x50	{10,20,31,42,52,82,95,106,114,116,127}
x2+x34	{4,13,21,36,45,66,85,98,100,109,117}	1+x50+x114	{10,18,20,42,52,63,82,95,106,116,127}
1+x2+x98	{4,13,21,34,36,45,66,85,100,109,117}	x82	{10,18,20,31,42,52,95,106,114,116,127}
x66	{4,13,21,34,36,45,85,98,100,109,117}	1+x114	{13,18,45,53,55,77,85,87,109,117,119}
1+x3	{2,6,13,34,38,45,66,74,99,102,109}	1+x19+x115	{18,21,39,50,51,53,71,83,103,114,117}
x3+x35	{0,22,30,62,64,67,86,96,99,118,126}	x51	{3,4,24,56,67,68,83,99,100,115,120}
1+x35+x99	{0,3,22,30,62,64,67,86,96,118,126}	1+x51+x115	{18,19,21,39,50,53,71,83,103,114,117}
x67	{3,26,27,30,62,90,91,99,122,123,126}	x83	{18,21,39,50,51,53,71,103,114,115,117}
1+x4	{10,13,15,42,45,74,77,79,100,106,111}	1+x20+x116	{12,17,24,44,49,52,56,84,88,108,113}
x36	{0,16,21,32,48,53,68,80,96,100,117}	x52	{6,14,38,46,53,84,85,102,110,116,117}
1+x36+x100	{0,2,4,8,34,40,64,68,96,98,104}	1+x52+x116	{12,17,20,44,49,56,84,88,108,113,120}
1+x68	{0,2,4,8,34,36,40,64,96,98,104}	1+x84	{12,17,20,44,49,52,56,88,108,113,120}
x5+x37	{10,20,31,42,52,69,95,101,106,116,127}	1+x21+x117	{10,17,49,53,54,74,85,86,106,113,118}
1+x5+x101	{2,7,20,34,37,39,52,66,69,103,116}	1+x53	{6,10,21,27,38,59,74,85,102,106,123}

Conclusions:

- Cube attacks seem to be ideal generic tools in leakage attacks
- They have the unique property that they can be applied even to poorly understood types of leakage from unknown implementations of unknown cryptosystems
- By using their robust version, they can be applied even when most of the measurements are known to be unreliable