Simple Power Analysis Threat in Embedded Devices?

Xi Xi Chen and Cathy Gebotys

Department of Electrical and Computer Engineering
University of Waterloo, Waterloo Ontario Canada
{xxchen, cgebotys} @ uwaterloo.ca

ABSTRACT

Many attacks against encryption algorithms recover the hamming weight information of the secret key; however, they fail to mention how to convert this information to the actual key bits. One way to do this is the brute force method and for the DES algorithm, requires 2^{45} tries before the correct key is found. Another method suggested by Biham and Shamir for DES uses linear equations to solve this problem. This paper specifically evaluates for the first time the likelihood of finding the correct key using this linear equations method given some theoretical values of measurement errors. The hamming weight information and actual measurement errors are found by measuring current consumptions of the DES key scheduling algorithm running on a 32-bit ARM processor commonly used in many embedded systems. Although linear equations are an effective method to convert hamming weight information into key bits, current readings from the ARM processor indicate that current and hamming weight does not have a direct relationship. Therefore, for the ARM processor, extracting hamming weight information from the change in current due to manipulating a round key is not possible.

Keywords

DES, Hamming Weight Measurements, SPA, embedded systems

1. INTRODUCTION

Cryptography is implemented in many embedded systems. Information leaks from these embedded systems is a major concern not only for the smartcard industry, but also for the wireless communications industry where PDAs, cell phones, laptops, etc must be secure from tampering and side channel attacks. Two well known attacks involve power or EM analysis. Here the power of the embedded processor is measured as it computes the cryptographic algorithm, and information from the power trace is used to obtain the secret key. In many situations only one power trace may be available, making simple power analysis (SPA) techniques threatening. In fact researchers have shown that hamming weights of key data is highly correlated to the power corresponding to key accesses in the program. Hence an important threat to cryptography in embedded systems is simple power analysis of single power traces revealing secret keys.

This paper forms an integer linear programming (ILP) model from the key-scheduling algorithm of DES. Each linear equation in the model is associated with a hamming weight value and small measurement errors to create the simplest case. Actual values of these measurement errors are determined from actual power traces using SPA techniques. The results of the paper demonstrated the effect of measurement error on the likelihood of finding the correct key. The paper is divided into five sections. Section 2 briefly lists previous SPA research. Section 3 formally defines the problem and outlines the ILP model. Section 4 evaluates how the number of keys to try is effected by measurement errors. Section 5 and 6 uses real power measurements from the test program and the DES key scheduling algorithm respectively to find the actual measurement error. Once the actual values of measurement errors are obtained, we can estimate the effectiveness of using this ILP model as an attack to the encryption algorithm.

2. PREVIOUS RESEARCH

In a private-key encryption algorithm like DES, the easiest way to obtain information regarding the round keys is to measuring the hamming weight of each round key during the key-scheduling function or when the round keys are used in the encryption function. Many research papers devise an attack by finding the hamming weights of the round keys. However, they neglect to mention how to find the round keys from the hamming weights.

An example of this can be seen in the timing analysis attack described in [1]. In this attack, the author used time differentials to find the hamming weight of the secret key of two different DES implementations. However, they failed to specify how to convert this hamming weight information to actual key bits.

In [2,8], researchers found that power consumption patterns of simple instruction have direct correlation to the hamming weight of the data being moved. Therefore, these power consumption traces can be easily converted to hamming weights. In [11], the researchers showed that power consumption patterns are not related to hamming weights. One can only assume that this relationship is based on devices used. The results in all three papers are obtained from different 8-bit smart cards and neither used power consumption of an encryption algorithm to confirm this relationship nor illustrated how SPA attack can be accomplished.

Once hamming weight is obtained from power traces, it can be converted to key bits using brute force. However, according to research by Messerges, Dabbish, and Sloan in [3], the brute-force search space for the DES key knowing the hamming weights is about 2^{45}.

A paper by Biham and Shamir in [4] suggested using linear equations to convert hamming weight information into key bit values. This method seems only possible for DES and is the only other method suggested besides brute force for DES.

Other published papers such as in [5] mentioned the use of linear equations; however, no paper including [4] had actually used this method or evaluated the effect of measurement error on the likelihood of finding a solution.

Therefore, the motivation driving this paper is to determine the likelihood of finding the correct solution using linear equations under the assumption of measurement errors. DES is used in this paper because it is still the most widely used private-key algorithm. Method used for DES can be easily converted to AES at a later date.

3. PROBLEM DEFINITION

The effectiveness of using linear equations to convert from hamming weight information to key bits assuming measurement errors is the major focus of this paper.

Specifically, the paper will look at the problem of how the number of possible keys to try will change with respect to specific hamming weight errors, the total number of measurement errors and change in the number of linear equations. When converting hamming weight into key bits, the measurement error is assumed to be at most one as this creates the simplest case. The actual measurement error is found using real power measurements from existing embedded processor such as the ARM in section 5 and 6. This processor is different than smart cards because it has a 32-bit bus and operates at a higher frequency.

3.1 Integer Linear Programming

We use Integer Linear Programming (ILP) model to convert hamming weights into key bits. To model an encryption algorithm, variables in the model would be the master key and the constraints would be equations representing the hamming weight of the round keys measured. The purpose of the ILP model in this case is to find all possible values of the master key and an objective function is not needed.

The ILP model is solved using General Algebraic Modeling System (GAMS) [12]. To find all solutions of a particular set of linear equations, the GAMS Corporation suggested using a method they called ICUT [5]. After each optimal solution is found, the model adds the following equations to the set:

$$\sum_i \left| x(i) - xsol(i,k) \right| \geq 1 \qquad (1)$$

Where 'i' is the index to the variable we are solving and 'k' is the index for each optimal solution. To make sure that the new set of optimal values x(i) hasn't already been found, we compare it with the previous set of x(i) values (xsol(i,k)). If they are the same, equation (1) is not satisfied, therefore, this new set of x(i) values is cut off or not included as the final set of integer solutions. Equation one is non linear due to the absolute function, however, it can be modified to become linear illustrated in [13].

3.2 Optimization model for the DES key scheduling algorithm

The linear equations used in this paper are the mathematical representation of the key scheduling function in DES. In the key scheduling function of DES, the master key is put through predefined permutation functions and shift functions. The bit value of the master key is maintained in each round key, but the bit positions have changed. Therefore, it is very easy to represent each round key by bits of the master key. To accomplish this in the paper, the key scheduling algorithm was rewritten in C, but the binary values of the master key were replaced by the position of the bits. Specifically, the master key variable was declared as an array of integer of size 56 and the values of this variable were assigned such that Mkey[0] = 0, Mkey[1] = 1…Mkey[56] = 56. The value of the array was manipulated the same way as the key scheduling algorithm and at the end of the algorithm, the value of each round key contained the position of the master key bits. For example, for round key 1, Rkey1[1..5] = {10,51,34,60,49} where the 10th bit of the master key is now the first bit of round key 1.

If the DES algorithm was run on an 8-bit smart-card, the hamming weight measured would correspond to the added result of each 8 bits of a round key. Since there are 48-bits in each round key, the round key can be separated into six groups of 8-bits that can be measured for its hamming weight. The first three groups of 8-bits define 28 bits of the master key while the last three groups of 8-bits define another 28 bits of the master key. Since each 28-bit set is independent, two models are defined for each set in this paper. Model one contain all constraints and variables used to define the first set of 28-bits of the master key and model two contain all constraints and variables used to define the second set. In each model variable X_i represent bit value of the master key and the index i defines the bit position. Bits 9 to 16 of each round key are used to form 16 constraints for model one and bits 25 to 32 of each round key are used to form 16 constraints for model two. An example of the first two constraints for model one is shown below

$$X_2 + X_9 + X_{19} + X_{42} + X_3 + X_{35} + X_{26} + X_{25} = HW_1 \qquad (2)$$
$$X_{59} + X_1 + X_{11} + X_{34} + X_{60} + X_{27} + X_{18} + X_{17} = HW_2 \qquad (3)$$

Variable X_2 would represent the bit value of the second bit of the master key and HWj is the measured hamming weight from the 8-bit bus. The HW_j values are calculated based on a random master key 0x10CB53314699756C.

In the more complex case we assume the algorithm runs on a device with 32-bit bus. Similar to the 8-bit bus, we take the first 32 values of each round key; add them together and the result is equal to the measurements from the bus.

3.2.1 Optimization model with measurement errors

The DES model formed so far does not take into account the effect of hamming weight measurement errors. In order to include the effect of measurement errors, an initial test was done using model one to find the number of integer solution of x_i given the correct hamming weight. Since model one had 28 variables and only 16 equations, one would expect more than one set of integer solutions for these 28 variables. However, the result from this test showed that there is only one solution for the correct set of hamming weights. After this test, the hamming weights (right hand side (RHS)) values of each equation in the model are changed randomly to reflect the fact that they could be erroneous. The number of integer solutions for each changed set of RHS values remained at most two. The result of this analysis is shown in Table 1. Each row in column one displays a different set of 4 equations that have wrong hamming weight values. This information is for analysis purposes only and will not be available to the attacker. In column 2, the largest number of solutions in any one set of 4 incorrect RHS values is 2. Adding each row in column two, the total number of errors for three sets of 4 wrong

RHS values is 5. This means that there are at least 5 different master key values for any set of 4 incorrect RHS values. Since one set of incorrect RHS values will almost always produce one value of the master key variable, it is more representative to find the number of possible sets of Z incorrect RHS values giving a specific value of Z.

Table 1: Number of solutions per fixed set of wrong equations

Equations with measurement Errors	Number of Solutions	Total number of errors
None	1	0
E1,E9,E7,E13	1	4
E5,E9,E11,E12	1	4
E11,E7,E12,E15	2	4
E5,E10,E11,E15	1	4

To include measurement errors, an extra variable err_j is introduced and this variable represents the hamming weight measurement error. For the simplest case, the value of this variable is binary. The RHS of each constraint is represented by $HW_j + err_j$ and an additional constraint $\sum_{j=1}^{16} err_j = Z$ is added to allow the user to specify Z. Model one will now solve for all solutions of err_j given the sixteen constraints and the number of measurement error (value of Z). For each solution of err_j, we assume there is one master key variable to verify.

The DES ILP model may be used in different situations due to the degree of information obtained by the attacker. In the first situation, the attacker knows exactly which hamming weight contain measurement errors, but not the magnitude of the error. The attacker can insert an err_j variable to only those hamming weights and set an appropriate bound for err_j to solve for a list of possible master key values to verify. The bound may be increased if the master key is not found. In the second situation, the attacker knows the number of measurements containing error, but not the specific hamming weights. The attacker can find a list of master key values by setting the value of Z and a bound for err_j to allow the ILP solver to find all solutions of err_j, denoted as NS_z. The third situation arises when the attacker does not know how many measurement errors exist or which hamming weights are erroneous. To find a list of possible master key values, the attacker must find each value of NS_z starting at Z = 1 until the correct key is found. Most of the experiments in section 4 assumes the third situation and the list of possible master key or err_j values is denoted as TS_n where $TS_n = \sum_{z=1}^{n} NS_z$.

4. EXPERIMENTAL RESULTS OF RUNNING ILP MODELS

4.1 Assumptions

We assume that there at most two values of x(i) for one value of err_j. We assume that for most values of err_j there are only one value of x(i). Therefore, the number of solutions presented is a close estimate of the actual number of solutions.

4.2 Initial Analysis

The two models assuming an 8-bit device was used in this section as the number of keys to verify and the execution time is much less than that of a 32-bit device. The purpose of this section is to gain an initial understanding of the relationship between the number of measurement errors and number of integer solutions (keys to try). To understand this relationship, all HW_j values are decreased by 1 to find all possible solutions of err_j for each value of Z. For example, if Z = 3, then it means that only three equations have the correct hamming weight value and 13 equations have measurement errors. Therefore, for this analysis, Z will not represent the number of measurement errors, but the number of correct hamming weight values. The result of this analysis is shown in Figure 1.

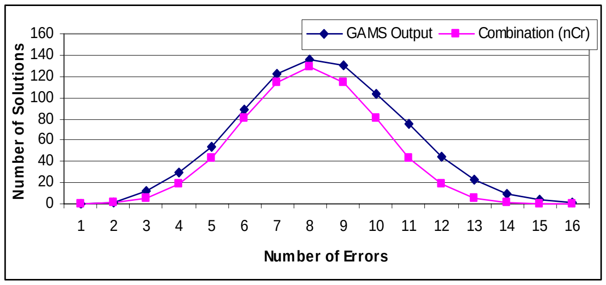

Figure 1: Solution Space Curve

In this figure, the solution space curve shows the general relationship between the number of solutions and the number of measurement errors. In fact, the most number of solutions occurs when half of the equations are correct and half are wrong. This relationship is the same as that of the combination function in probability theory, which is also shown in this figure. Essentially, the number of integer solutions is a subset of the number of combinations of "r" chooses "n" where "r" is equivalent to Z and "n" is 16 in this case. The combination function curve has been scaled down to give the reader the comparison between the two curves.

Looking at figure 1, two conclusions can be drawn. If the attacker has a good estimate on the number of measurement errors made, he or she could use various methods to avoid having 7, 8, or 9 measurement errors as this will create a large number of keys to verify. If the attacker can not make this estimate, figure 2 implies the following relationship.

$$TS_6 = \sum_{z=1}^{6} NS_z, TS_{12} = \sum_{z=1}^{12} NS_z \text{ then } TS_6 < TS_{12}$$

TS_n is the total number of solutions for 'n' number of erroneous equations and NS_z is the number of solutions for each value of Z where $Z \leq n$. Erroneous equations are constraints in the model with a hamming weight error.

In general, the greater the number of measurement errors the larger the number of keys to verify. However, there are exceptions to this case discussed further in section 4.3.

4.3 Critical Equations

In an actual attack, the attacker would not know which of the hamming weight measurements are correct and which are not. The correct hamming weight will stay the same, but the incorrect hamming weights will be one less than its correct value. Both correct and incorrect measurements will become the RHS of each equation. The equations of model one containing measurement errors are shown in the "Equation Change" column in table 2. The RHS of these equations are decreased by one to reflect the measurement error. The total number of solution (TS_n) for each set of measurement errors is shown in the "# of Solution" column. The equations that are underlined are the "Critical Equations", which means that if these equations were erroneous, the total number of integer solutions would increase dramatically. Each equation (constraint) is labeled E1 to E16 where E1 and E16 consist of bits of the first and sixteenth round key respectively.

Table 2: Analysis of the effect of errors on the number of solutions

Equation Change	# of Solutions	Equation Change	# of Solutions
Total Error = 2		**Total Error = 8**	
E1, E9	19	E1,E9,E10,E11,E8, E4,E5,E13	1016
E7, E16	1	E9,E10,E11,E8, E4,E5, E13,E15	522
E9,E10	23	E9,E10, E12,E3 E4,E5,E13, E15	24
E9,E16	6		
Total Error = 5		**Total Error = 13**	
E1,E9,E10,E11,E8	565	E1,E9,E10,E11,E8,E4, E5,E13,E15,E12,E3, E2, E16	879
E1,E9,E11,E8,E4	177	E9,E10,E11,E8,E4,E5, E13, E15, E12, E3, E2,E16,E6	181
E1,E9,E4,E5,E13	37	E9,E10,E4,E5,E13, E15,E12,E3,E2,E16, E6,E7,E14	19

Looking at table 2, row 3, if the RHS of critical equations E1 and E9 are erroneous, the number of master key's to verify is around 19. For critical equations E9 and E10, the number of solutions is slightly more than 19. However, for non-critical equations E7 and E16, the number of solutions is only 1. Similarly, when there is only one critical equation E9, the number of solutions is less than 19, but greater than 1. The same patter is repeated for total error n = 5, 8 and 13. In most cases, when one critical equation is taken away, the number of solutions decreases by almost 50%. Therefore, critical equations play an important role in the number of solutions the attacker has to try. If the attacker has knowledge of which equations are critical, he or she can be extra careful when making measurements to the RHS of these equations and the number of solutions would decrease significantly.

Figure 2 implies that $TS_i < TS_j$ for $i<j$. However, one can see table 2 that $TS_8 > TS_{13}$ if the same 5 critical equations were erroneous. If the solution space curve for n = 8 and 13 were plotted, the graph for n = 8 would be a vertical up shift of the graph for n = 13. Also, the minimum value of "z" that makes this set of RHS values feasible for error of 13 is higher than error of 8. Therefore, fewer z values are added together to formulate TS_{13}. In conclusion, the number of solutions is greatly affected by the critical equations and not so much by the total number of errors.

4.4 Critical Equation Identification Method

Since critical equations contribute to large number of solutions, finding a way to identify them is essential. The only way to identify these equations without using the trial and error method is to evaluate the effects produced by one equation at a time. If the TS_n value is obtained from a set of RHS values assuming one erroneous equation, the erroneous equation causing the largest TS_n value would be critical. The results of this experiment done on model one is shown in figure 2 and equations that are circled are confirmed by the trial and error method.

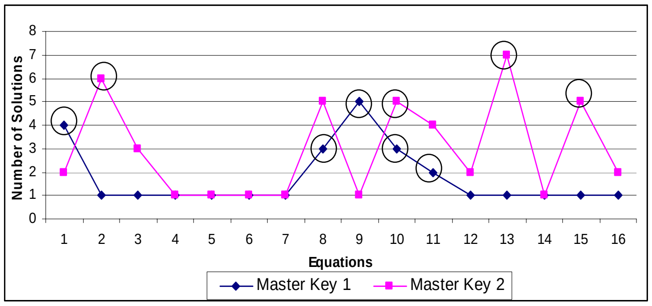

Figure 2 Critical Equation Identification for two master keys

For the master key defined in section 3.2 (master key 1), the critical equations are equations 1, 8, 9, 10 and 11. For a different master key (master key 2), the critical equations are completely different. Therefore, critical equations are dependent upon the value of the master key. The only way to determine them is to know the actual hamming weight values.

4.5 Worst Case Analysis for 8-bit bus

This section will find a specific set of erroneous equations that will give the largest TS_n value for models one and two defined in section 3.2. The worst-case analysis is only an approximation as different combination of non-critical equations may result in a slight change in the value of TS_n as seen in Table 2.

To find the set of erroneous equations in the worst-case, critical equations were identified first. If the number of critical equations is less than half the number of total equations, add random equations to the erroneous set until the number of equations in this set is half the number of total equations used for the model. Due to the shape of the solution space, if the value of Z is equal to half the number of total equations, then NS_z is the largest for all values of Z. Therefore, at least this many erroneous equations must be included to produce large number of master keys to verify.

After half the number of total equations is included, a random equation is added to the set if the new TS_n value is greater than before. If the TS_n value is smaller, this random equation is discarded and a new one is chosen. If all equations are exhausted, then adding more equations will not yield more solutions and this set of erroneous equations will produce the largest TS_n value and is the worst case.

Instead of comparing the TS_n produced by each set of erroneous equations, we can compare NS_z when $Z = ½ K$ where K is the total number of equations in this model. Using the method described above, the combination of erroneous equations that give the largest TS_n value for model one is E1, E10, E8, E9, E11, E4, E12, E14, E7, E2, E16 and E13 where E1, E10, E8, E9 and E11 are critical equations. The set of worst-case erroneous equations for model two is E16, E3, E12, E8, E6, E4, E10, E2, E11, E13, E15 and E7 where there are no critical equations. The worst-case TS_n value is 2408 and 1040 for model one and two respectively. Since there are no critical equations in model two, the TS_n value is much less than model one. The final value for the total number of master key to verify is the product of the TS_n values of both models. This is due to the fact that each set of 28-bit value in the first model must be combined with every set of 28-bit value in the second model to form one value of 56-bits of the key. The largest number of master keys to verify due to measurement errors to obtain 56-bits of the master key is TS_{worst} = 2408 x 1040 = 2,504,320. The technique of forming two models defining 28-bits each, instead of one model defining all 56-bits of the master key reduces the execution time of the ILP solver without increasing the number of master keys to verify. If one model were used, the time taken to find all 2 million values would be very long whereas the execution time of each smaller model was in the order of seconds. Since the equations and variables of each model is independent, the TS_n value produced by one model defining 56-bits is the same as the product of the TS_n values produced by two models each defining 28-bits

This worst-case analysis in this section applies only to the master key defined in section 3.2 and assumes that the attacker measures only 32 hamming weights. Measuring more hamming weights and additing additional constraints to the model can reduce the number of solutions. The effect of additional equations on the number of solutions is shown in the next section.

4.6 Additional Equations

Additional constraints can be added to the model by measuring more 8-bit hamming weights from the device. For example, for model one, these constraints could come from bit 1 to 8 or 17 to 24 of each round key. By adding extra constraints to the model, cases where more than one set of x_i values exist for one set of err_j values will decrease because these new constraints will shrink the solution space.

The most important effect of additional constraints to the model is that the value of TS_n decreases for all combinations of erroneous equations. Figure 3 shows the solution space curve for 19 constraints versus 16 constraints in the model. It is clearly shown that the number of solutions for 19 constraints is much less than 16 constraints. The worst case scenario for model one with 19 constraints with model two unchanged is TS_{worst} = 1106 x 1040 = 1,150,240, which is much better than model one with 16 constraints.

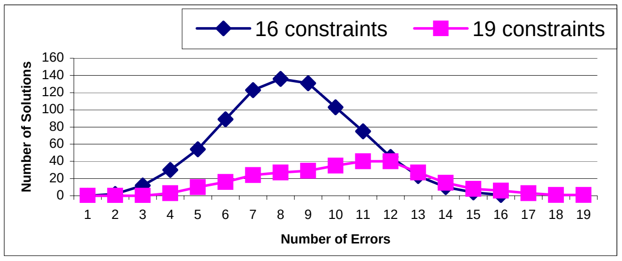

Figure 3: Solution Space curve for model with 19 equations versus 16 equations.

In general, additional equations will shrink the solution space of the set of possible master keys. However, these additional equations could also be erroneous and execution time of the ILP model may increase. The increase in time as a result of the additional equations was minuscule for models one and two.

4.7 32-bit Bus Analysis

This section is intended to convert model one and two into an ILP model for a 32-bit bus. When the DES key scheduling algorithm is run on a 32-bit bus, the most common way of storing round keys to memory via the bus is to store the first 32-bits of the round key, followed by the last 16-bits of the round key, implemented in [14]. In this case, the GAMS model will consists of two sets of 16 constraints with the first set compose of 32 variables and the second set compose of 16 variables. The RHS of each equation will represent the measured hamming weight values plus errors.

Once the constraints have been changed to accommodate a 32-bit bus, the relationship between master key variable and error variable must be re-established. If the one to one relationship between these two variables does not hold, another ILP model must be constructed to find the number of master key variable for every set of error variable values. If the last 11 equations with 16 variables are removed from the model, there are approximately 50-60 master key integer solutions per random set of err_j values. If the last 6 equations with 16 variables are removed, the resulting model has 16 constraints with 32 variables and 7 constraints with 16 variables. The one to one relationship holds for this model and the set of constraints in this model is the minimum number to ensure this relationship. Using the critical equations of this model as a set of erroneous equations, the process to solve for TS_5 caused the OSL solver to exceeded sources. If three critical equations are replaced with non-critical equations, the TS_5 value took two days to obtain and is equal to 1967. The TS_5 can be reduced by adding additional constraints to the model; however, the execution time to obtain the TS_n value for higher n takes days.

Even if the model had all 32 equations, the total number of solution assuming the worst case still takes days to obtain. It is essential to obtain a solution space curve by decreasing all hamming weights and changing Z to find the appropriate NS_z distribution. The execution time to obtain these values is also recorded. Table 3 shows the NS_z value and its execution time for some values of Z for the model with all 32 equations.

Table 3 NS_z values for some values of Z for 32-bit model

Z	NS_z	Time Taken†	Z	NS_z	Time Taken†
1 to 4	0	5s	24	755	18 hrs
5	2	5s	26	319	2.5 hrs
6	14	2min[1]	27	113	28 min
7	72	7min	28	24	15 min

† NS_z values for z =1-4, 12, 21, 24, 26-32 are obtained using GAMS solver LAMPS while z = 5, 6, 7, 8, 10, 16 are obtained using GAMS solver OSL.

8	247	11 hrs	29	16	7 min
10	1012	24 hrs	30	6	3 min
12	1022	26 hrs	31	3	1 min
16	1638	2 Days	32	1	5s
21	773	26 hrs			

The results of table 3 suggest that the execution time to obtain a list of possible master keys for a device with 32-bit bus is very long. Results from previous sections suggest that if critical equations were included in the set of erroneous equations, the NS_z value for each Z is much larger than the values shown in table 3. However, in [XX], we show that table 3 represents the average execution time and NS_z values and both can be estimated using this table.

In conclusion, the number of possible keys to verify for a 32-bit bus device is small compared to the brute force. However, the execution time may make the technique infeasible. In the next section of this paper, we will measure the actual hamming weight values to confirm the theoretical values of the measurement errors used in this section.

5. POWER MEASUREMENTS

This section of the paper will use real power measurements to find the actual measurement errors. The real power measurements will be obtained from a test program and the DES key scheduling algorithm. Specifically, we use an inductive probe, which wraps around the power line connecting the processor core to its supply. Therefore, the change in current is measured instead of power. The current consumption reflects the change in current in the ARM7TDMI processor with a 32-bit bus. The processor core has a 20MHz clock and 3.3v supply voltage. We use a Digital Phosphor Oscilloscope to measure the change in current of our inductor probe and a trigger implemented in the assembly program initiates measurements on the scope.

5.1 The Test Program

In our test program, we manually input the hamming weight value before each run and the program fetched 60 unique values of this hamming weight for our experiment. The code for our test program is shown below:

```
Define Hamming Weight (manual Change)
Loop start
    Trigger Code                  ; 1.  set the trigger for the scope
    100 x NOPS                    ;2.
    Address Calculation Code      ;3. load values into r0 & r4
    LDR    r7,[ r0, r4, LSL #2]   ;4. load HamWt value into r7
    MOV    r10,  #&00000000       ;5. move all zeros into r10
    100 x NOPS                    ;6.
    STR    r10,[sp,#0]            ;7. store zeros into memory
    100 x NOPS                    ;8. via the bus
    STR    r7,[sp,#4]             ; 9.store value of r7 into memory
    100 x NOPS                    ; 10. via bus
    LDR    r10,[sp,#0]            ;11. load zeros into memory
    100 x NOPS                    ;12. via bus
    LDR    r7,[sp,#4]             ;13. load value of r7 into memory
    400 x NOPS                    ;14. via bus
go to start
```

The first part of the assembly program consists of setting up a trigger value for the oscilloscope to measure the current consumption. In the original test program, line 3 is the code used to calculate the location of a value of hamming weight X from a predefined table in memory. Since the code is several lines and it is irrelevant to the experiment, it is not shown. Line 4 of the program moves the specific value of hamming weight X into register r7. Line 7 and 9 stores zeros and the actual value of hamming weight X into memory via the bus respectively. Line 11 loads all zeros from the stack position to which it was stored, and line 13 loads the actual value of hamming weight X from stack position to which it was stored. This whole program is run in an infinite loop. Each time the program is run, a different value of hamming weight X is loaded from the predefined table until all 60 values of that hamming weight have been loaded. At that point, the loaded values cycles back to the first value and the process continues indefinitely. Storing and loading all zeros onto the stack before the actual value act as a reference point, which will become apparent later. Each portion of the code is separated by 100-200 NOPS for signal settling purposes.

For each hamming weight 1500 current traces or frames. Each frame corresponds to each run of the program and contains 10,000 data points sampled at 250MSamples/s. We average all 1500 frames together to get the averaged current consumption trace (ACC_x) for that particular hamming weight x. All data analysis is done using Matlab and each frame is put into an array of size 1x10, 000.

Figure 4 illustrates the current consumption corresponding to each line of the test program. The load operations are much higher than the store operations and this trend was consistent in each run of the program. The initial current consumption curve was quite noisy, so we used a mean filter of size 100 to smooth the curve. Figure 4 shows the peak of current consumption corresponding to line 11 of our test program for hamming weight of 0, 16 and 32 after being filtered by the mean filter. Figure 6 shows the power consumption of line 13 of the test program for hamming weight of 0, 16 and 32.

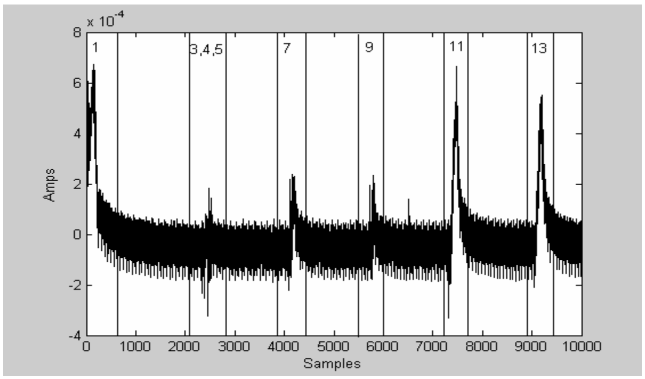

Figure 4: Current Consumption of Test Program

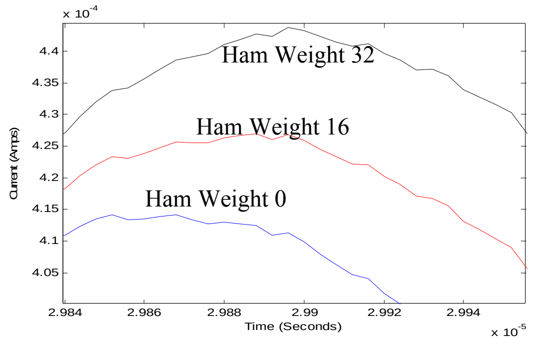

Figure 5: Current Consumption at Line 11 of test program for Ham Weight 0,16 and 32

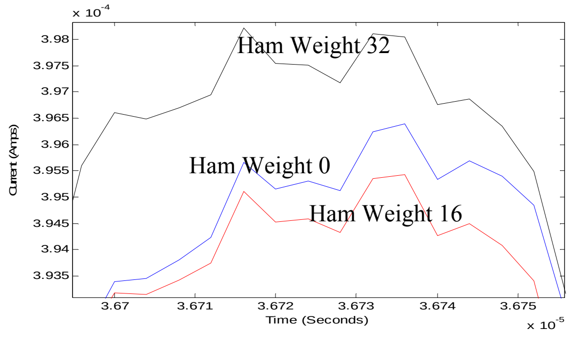

Figure 6: Current consumption at line 13 of test program for Ham Weight of 0, 16 and 32

From Figure 5 and 6 we can see that noise play a very important role in current consumption measurements. In Figure 5, the same hamming weight value was loaded in all three cases but the consumption was recorded at different times. As a result, the power consumption varied drastically. In Figure 6, we can see that the power consumption of loading hamming weight of 16 should be in between that of hamming weight of 32 and 0, but because of the shift shown in Figure 5, Hamming weight of 16 is lower than hamming weight of 0. Therefore, when we calculate the current consumption value for each hamming weight, we must take into account the shift shown in Figure 5.

5.2 Converting Current into hamming weight

To convert the current consumption values into hamming weight values, we must first make an assumption. We assume that the current consumption difference between any two adjacent hamming weights is only δ. To calculate δ, we find the difference in current consumption between hamming weight of 0 and 32 and divide that difference by the total number of hamming weights. To find the peak current consumption of each hamming weight, we take five consecutive values that best represent the peak of that hamming weight. For example, looking at Figure 6, we can't say that the current consumption of hamming weight 0 is at the highest current value because this point might have been caused by noise. Instead, we say that the peak current consumption of hamming weight 0 is the average of five values around the highest current value. Table 4 shows the current consumption of the processor core at line 11 of the test program for hamming weight of 0 and 32. The final current consumption for each hamming weight is the average of the values at the respective sample locations.

Table 4: Current consumption at line 11 of test program

Ham Weight 0		**Ham Weight 32**	
Time (μs)	Current(μA)	Time (μs)	Current(μA)
29.860	412.83	29.888	440.77
29.864	412.91	29.892	441.27
29.868	412.93	29.896	441.66
29.872	412.36	29.900	441.64
29.876	411.66	29.904	435.86
Final	**412.538**	**Final**	**440.24**

Table 5 shows the current consumption of the processor core at line 13 of the test program for hamming weight of 0 and 32.

Table 5: Current consumption at line 13 of test program

Ham Weight 0		**Ham Weight 32**	
Time (μs)	Current(μA)	Time (μs)	Current(μA)
36.716	394.62	36.716	396.73
36.720	394.17	36.720	396.82
36.724	394.45	36.724	396.22
36.728	394.45	36.728	396.31
36.732	394.24	36.732	396.09
Final	**394.386**	**Final**	**396.434**

Assuming that the current consumption of loading the same value into memory is the same from run to run, one can see from table 11 that current value for hamming weight 32 is shifted up by 440.24 – 412.538 = 27.702μA. The actual averaged peak value (αC_{32}) of loading a value with hamming weight of 32 into memory is 396.434 – 27.702 = 368.732μA. The δ value is calculated based on equation 4 where z is the smallest hamming weight and y is the largest hamming.

$$\delta = (\alpha C_z - \alpha C_y)/(y-z) \qquad (4)$$

For the set of current values in table 11 and 12, αC_0 =394.386μA and δ = (394.386 – 368.732) / 32 = 0.8016875μA. Once the δ value is obtained, it can be used to find hamming weights of other current consumption traces. In an actual attack, the calculation of δ will be accomplished on the attacker's own device and the current consumption of unknown hamming weights is measured from the attacked device.

Table 6 shows αC_x values at line 11 and 13 of the test program for hamming weight of 13, 14, 16 and 17.

Table 6: Current Consumption of Ham Weight 13, 14, 16, 17

Current Consumption at line 11			
Ham Weight	**Current (μA)**	**Ham Weight**	**Current (μA)**
13	402.782	14	407.644
16	425.018	17	419.734

Current Consumption at line 13			
Ham Weight	Current (μA)	Ham Weight	Current (μA)
13	374.928	14	378.654
16	393.618	17	387.916

If the attacker obtained two current consumption values 419.734μA and 387.916μA for loading zero and hamming weight X into memory respectively, they can convert these values into the hamming weight values by comparing them to the current consumption of hamming weight 0. Looking at Table 4, we can see that the current consumption of hamming weight X is shifted up by 419.734 – 412.538 = 7.196μA. Therefore, the actual current consumption of loading a value with hamming weight of X into memory is 387.916 – 7.196 = 380.72μA. To obtain the X, we can use the following formula

$$HW_x = (\alpha C_0 / \delta) - (\alpha C_x / \delta) \quad (5)$$

Using the two current consumption values, we have HW_x = (394.836/ δ) – (387.916/ δ) = 17.04654245. Looking at Table 6, the two current consumption values correspond to that of hamming weight 17.

Table 7 shows the calculated hamming weights using the current consumption values in Table 6. We can see that for these four values, the largest measurement error is 1 as we had originally predicted, and the error is an underestimate of the actual value.

Table 7: Calculated versus actual hamming weight values

Actual HW	Calculated HW	Actual HW	Calculated HW
14	13.51898339	16	16.52514228
13	12.03461449	17	17.04654245

The experiment outlined in this section was repeatable and each time, the calculated δ value was very close. With the results from the experiment so far, the measurement errors seem small enough for this attack to be effective. In the next section, we apply the method devised in section 5 to the actual key-scheduling algorithm of DES.

6. POWER MEASUREMENTS FROM DES

6.1 Round Key Identification

Devices implementing the DES key-scheduling algorithm usually offer a function to perform the full DES key expansion. In the case of embedded devices, there is usually enough memory for the device to pre-calculate and store all round keys before the encryption is performed. The key scheduling function has a distinct current consumption characteristic than the en or decryption function; therefore it is easily identified when current consumption measurements are taken. The DES key-scheduling algorithm is extremely repetitive and identifying each round of the algorithm is very easy. Figure 7 shows the unfiltered averaged current trace of the DES key-scheduling algorithm. The C code for this particular key scheduling function is an optimized version of the one found in [15] where software optimization techniques were applied to reduce the execution time.

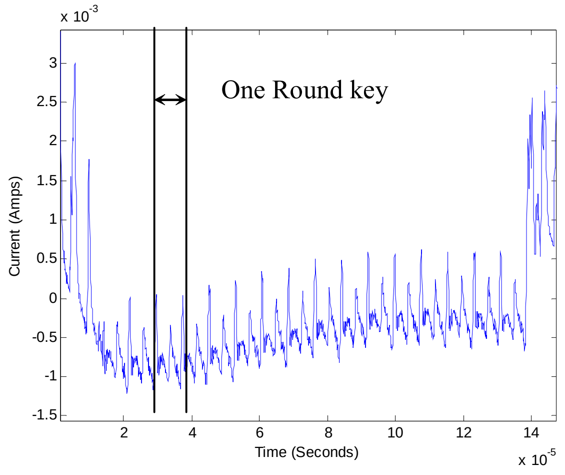

Figure 7: Current Consumption trace of the DES key scheduling algorithm

Figure 7 shows that regardless of the C implementation, the basic assembly operation for the permutation function will consists of shifts and AND operations to move bits to certain positions. These instructions require much less current than the store operation that put round keys into memory. The pair of one short, one long spike thus corresponds to one round key.

Once the round key region is established, the attacker can adjust the scale to capture the current consumption trace related to just one round key. Figure 8 shows the zoomed-in unfiltered current consumption trace of round key one

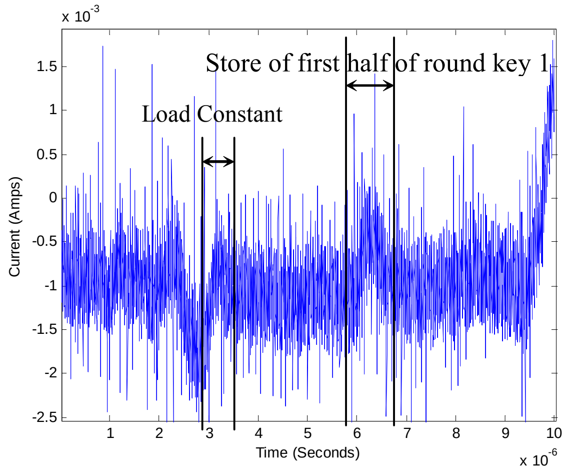

Figure 8: Current trace of storing the first half of key 1

If the attacker has some information regarding the implementation of the algorithm, he or she can match the bumps with the number of load and store operations in the code to identify the exact point of round key manipulation. For testing purposes, one can also insert NOPs before and after the store operation in the assembly file or use DPA to confirm the location of the store operation. If the attacker has no information regarding the implementation and cannot insert or delete code in the algorithm, he or she can make an educated guess based on the current consumption trace. The constant current before and after the spike in figure 8 must correspond to the shift, AND and rotate operations used to calculate each half of the round key values and the spike must be the operation to store one part of the round key to memory.

6.2 Convert Current into Hamming Weight

To apply the method introduced in section 6 to convert current traces into hamming weights, the attacker must know the implementation detail of the algorithm. That is because we require a reference point to shift all current traces so that a direct relationship exists between current traces and hamming weights. By comparing the assembly code of the key scheduling algorithm and the current trace, we found a load operation "ldr r12, |L1.1004|" that occurs before the store of the round key and there are no other memory access operations between the two. Therefore, this peak, also shown in figure 8, is used as a reference point to shift all current traces. The filtered, shifted current consumption trace of storing ten unique values of hamming weight 0, 1, 10, 12, 15, 20 and 24 is shown in figure 9. Each trace in figure 9 is the average of 1500 individual current traces.

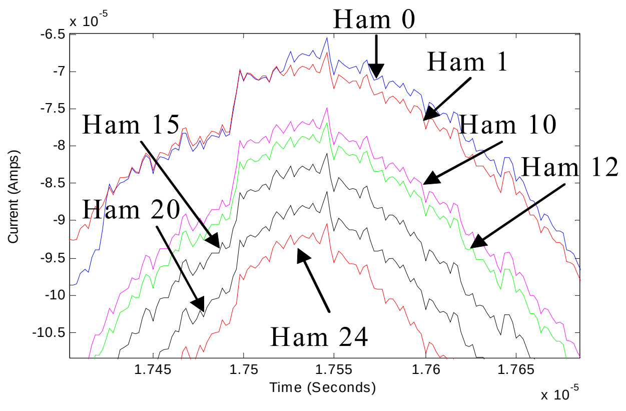

Figure 9: Averaged current traces of storing first half of first round key into memory

The results from figure 9 show that the hamming weight order in the key-scheduling algorithm is maintained when the respective traces are aligned. Once the hamming weight order has been established, the attacker can find the δ value from the averaged traces of hamming weight 0 and 24. Table 8 shows averaged current consumption peak value (αC_x) at the peak of the operation used for alignment (LDR const) and the store operation that puts first half of the round key one on the bus (STR rdkey).

Table 8: αC_x value of hamming weight 0 and 24 used in the δ calculation

Current Consumption at load const			
Ham Weight	**Current (μA)**	**Ham Weight**	**Current (μA)**
0	**-145.334**	24	**-169.69**
Current Consumption at store round key			
Ham Weight	**Current (μA)**	**Ham Weight**	**Current (μA)**
0	**-67.9374**	24	**-117.138**

Using the averaged current value at the load instruction as a reference, one can see that the αC_{24} has been shifted up by -145.334+169.69 = 24.356μA and the actual αC_{24} value is -117.138+24.356 = -92.782μA. Using equation 4, the δ value is calculated as δ = (-67.9374+92.782)/24 = 1.035191μA. Once the δ value is found, the attacker can obtain current traces from the attacked device to find its hamming weight.

Table 9 shows the final averaged peak value (αC_x) values from shifted averaged current traces consisting of 1500 CC traces as well as the actual hamming weight. Using equation 5, the calculated hamming weight is also shown in table 18. The actual hamming weight is used for comparison and is not available to the attacker.

Table 9: Calculated Hamming weight using the Delta Method

αC_x (μA)	Calculated Ham Weight	Actual Ham Weight
-69.789	1.7891	1
-77.273	9.0186	10
-79.195	10.87591	12
-83.420	14.9574	15
-88.602	19.9644	20

Results from table 9 shows that mathematically, the averaged current consumption value of each hamming weight is very consistent and the calculated hamming weight value is at most one less than the actual hamming weight with the exception of hamming weight one. The results of this section were repeatable and each time the δ value was very close.

7. CONCLUSION

In this paper, the effectiveness of using linear equations to convert from hamming weight information to key bits of DES assuming measurement errors was analyzed. An ILP model was devised for this conversion and the measurement errors were set to the simplest case. Using a simple 8-bit bus model in the theoretical part of the paper, we were able to evaluate how the number of keys to try was affected by measurement errors. Combinations of critical equations gave large number of keys to try than combination of non-critical equations. In general, more measurement errors mean more possible keys to try. However, this paper showed that specific combination of equations play a more important role in determining the number keys than the number of measurement errors. When the number of linear equations increased, the number of integer solutions decreased significantly. The same trend between measurement errors and number of keys can be applied assuming a 32-bit bus. However, the execution time of the 32-bit model may be too long to be feasible.

The actual measurement error was found using current consumption traces of a test program and the DES algorithm running on the ARM7TDMI processor core. The averaged current consumption traces of hamming weight X is the average of several values of that hamming weight. This averaged trace provided good results, however the standard deviations were large, reflecting the fact that the spread of values of the same hamming weight was large. Therefore extracting hamming weight of a single round key from a single or several current traces was not successful for this processor. A possible explanation for this spread is that different bit lines contribute differently to the current in the ARM processor and this is shown in [9].

All research in this paper focused on the DES algorithm. However, ILP techniques can be applied to the AES algorithm to convert hamming weight into key bits and is shown in [9]. In conclusion, the ILP technique is very effective in converting hamming weight information into key bits. However, due to different current contributions of different bit lines, hamming weight and its current consumption does not have a direct relationship. Thus, it is difficult to extract the hamming weight information of a single round key on the ARM7TDMI processor core. Future work will examine the bit contributions to linearize the relationship between current and hamming weight as well as examining information leaked from EM readings.

8. ACKNOWLEDGEMENT

This research was funded in part by grants from CITO, NSERC and RIM

9. REFERENCES

[1] A. Hevia, M.Kiwi, Strength of Two Data Encryption Standard Implementations Under Timing Attack, Proceedings of LATIN'98, Lecture Notes in Computer Science Vol.1380, pp. 192-205, Springer-Verlag, 1998

[2] R. Mayer-Sommer, Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards, Proceedings in CHES 2000, Lecture Notes in Computer Science Vol.1965, pp.78-92, Springer-Verlag, 2000

[3] T.S.Messerges, E.A.Dabbish, R.H.SLoan, Investigation of Power Analysis Attacks on Smartcards, Proceedings of the USENIX workshop on Smartcard Technology, pp.151-161, 1999.

[4] E. Biham, A. Shamir, Power Analysis of Key Scheduling of the AES Candidates, Second Advanced Encryption Standard Candidate Conference, Rome, 1999

[5] T. Lash, A Study of Power Analysis and the Advanced Encryption Standard, MS Scholarly Paper, George Mason University, Feb. 2002.

[6] GAMS Corporation, icut.gms Integer Cut Example, http://www.gams.com/modlib/libhtml/icut.htm,

[7] P. Kocher, J. Jaffe, and B. Jun, Differential Power Analysis, Advances in Cryptology Proceedings of CRYPTO '99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, 1999.

[8] T.S. Messerges, E.A.Dabbish, R.H.Sloan, Examining Smart-Card Security under the Threat of Power Anlaysis Attacks, IEEE Transactions on Computers Vol. 51, May 2002

[9] X. Chen, Simple Power Analysis Threat in Embedded Devices, Master's Thesis, University of Waterloo, 2004

[10] National Bureau of Standards (U.S.), Data Encryption Standard, Federal Information Processing Standards publication (FIPS PUB 46-3), October 1999

[11] M. Akkar, R. Bevan, P. Dischamp, et al. Power Analysis, what is now possible…Proceedings in *ASIANCRYPT2000*, *Lecture Notes in Computer Science*, Vol. 1976 of, pp. 489-502, Springer-Verlag, 2000

[12] A. Brooke, D. Kendrick, A. Meeraus and R. Raman, GAMS: A User's Guide, GAMS Development Corporation, 1998, Available at http://www.gams.com/docs/gams/GAMSUsersGuide.pdf

[13] E. Kalvelagen, Integer Programming Models 1: Selected Models, GAMS Corporation, Available at http://www.gams.com/~erwin/mip1.pdf, pp.31, 2001

[14] K. Loudon, Master Algorithms with C, O'Reily, 1999

[15] P. Karn, DES/3DES code in portable C, available at http://www.ka9q.net/code/des/, 2000