This section will start with a review of three commonly used encryption algorithms. It will then move into descriptions of three types of power analysis. Following that, several defences against power analysis will be discussed. It will conclude with a description of a defence implementation on an FPGA.

*A. Standard Encryption Algorithms*

There are three main encryption algorithms this paper is going to touch on, Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Rivest-Shamir-Adleman (RSA). DES and AES are private key symmetric block ciphers and RSA is an asymmetric public key cipher. DES and RSA will be briefly mentioned, but the understanding of these algorithms will not be vital to the rest of the paper. The AES algorithm will be the focus of my research in following sections.

*i. Data Encryption Standard*

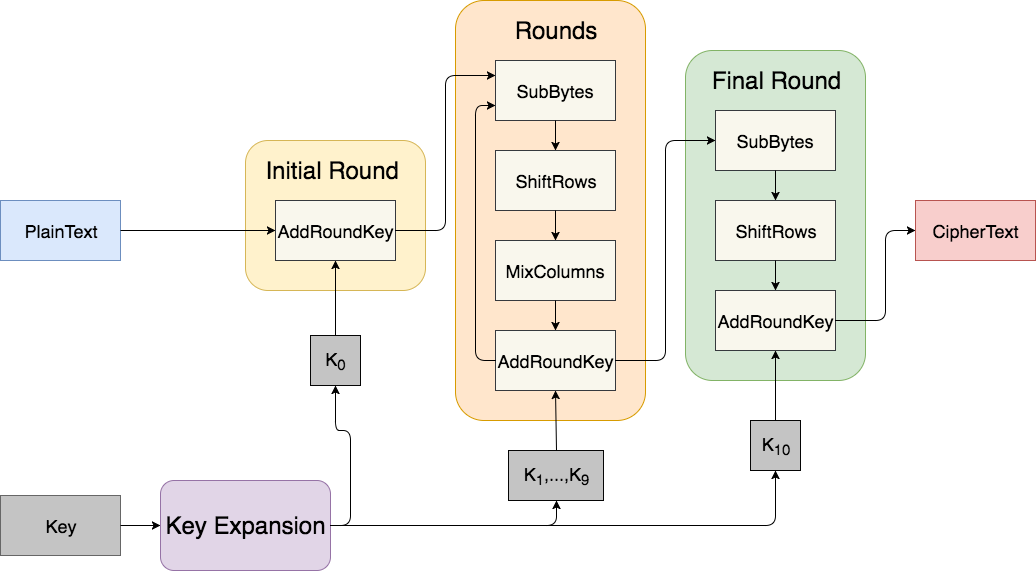
DES is an encryption standard block cipher that takes a 56-bit key and 64-bit plaintext block and outputs a 64-bit ciphertext block. DES is based on the idea of confusion and diffusion [3]. The SBox lookup tables are responsible for the confusion, which produce an output that has no visible relationship to the input. Diffusion is accomplished by multiple permutations and bitwise expansions to eliminate any repetitiveness that is present in the plaintext.

DES is a symmetric encryption algorithm. This means the same key is used to encrypt and decrypt data. The only modification in decryption is the round keys are issued in reverse order. Each block takes sixteen rounds to encrypt and in these rounds the plaintext is shifted, permuted, expanded, condensed, and substituted to achieve an output that has no statistical correlation to the inputted data [4].

This algorithm was the first official standard cipher used in the commercial world, but as technology has advanced, brute force techniques have been proven to break this algorithm. The vulnerability in DES is that the algorithm only has a 56-bit key and, as the speed of computers has increased, it is now possible to run through all possible values of the key through simple guess and check [5]. A variation of DES has been developed called 3DES that uses the same algorithm but loops through it three times. This requires a 168 bit key, which is unbreakable through brute force techniques. The downside of using 3DES is that it is very computationally heavy. DES has to run three times for the data to be fully encrypted and this is often too much overhead for a device. This is where the motivation behind developing a new encryption standard, the Advanced Encryption Standard, comes from.

*ii. Advanced Encryption Standard*

AES, similar to DES, is a symmetric block cipher that was first standardized in 2002 [6]. AES operates on 128 bit blocks and offers an option of a 128, 192, or 256 bit key. There are four separate phases of AES, Key Expansion, Initial Round, Rounds, and Final Round. The following explanation assumes a key size of 128 bits, or 16 bytes, and will be looking at the encryption of a single block of 128 bits of plaintext. This process is close to identical to AES with 192-bit or 256-bit keys with some minor variations, including the addition of rounds and a modified Key Expansion phase. Figure 1 shows an overall representation of the AES rounds and dataflow between stages, which is discussed in [6].

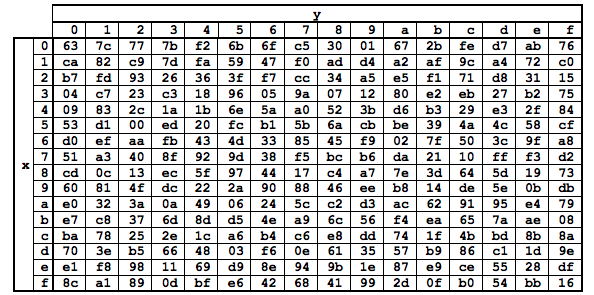


**Figure 1:** General overview of the AES encryption algorithm. The arrows represent the data flow through the algorithm. Each stage takes in 128-bit blocks and outputs 128-bit blocks. The Initial Round phase executes once, the Rounds phase executes nine times, and the Final Round phase executes once.

The first phase of AES is the Key Expansion. The role of Key Expansion is to map the private encryption key to multiple round keys, each 128 bits. There needs to be eleven round keys generated, nine for the Rounds phase, one for the Initial Round phase, and one for the Final Round phase. Key Expansion is done by first copying the initial 16 bytes of the original key into the first 16 bytes of the expanded key. Next, the least significant four bytes are rotated, substituted using a S-box, and then XORed with a value from the lookup table. The value from the lookup table is depended on the current iteration the algorithm is in. These four bytes are then XORed with the bytes 16 positions to the left of it and added to the end of the expanded key. This process is repeated until the entire expanded key is produced.

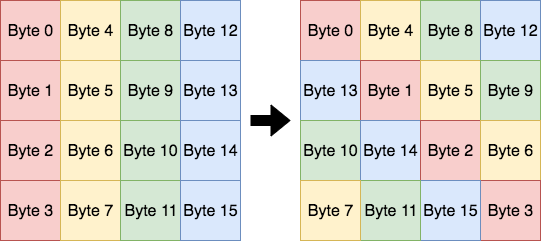
The next phase is the Initial Round phase. The only stage that occurs in this phase is AddRoundKey. In this stage the inputted plaintext is XORed with the a round key, K0. The output of this phase is fed into the Rounds phase.

The Rounds phase has four different stages in it, SubBytes, ShiftRows, MixColumns, and AddRoundKey. The stages are repeated nine times before moving onto the Final Round phase. SubBytes consists of inputting the bytes into an S-box lookup table where they are substituted by their corresponding table values. Figure 2 shows a representation of the mappings of inputs of the S-Box to the outputs of the S-Box.



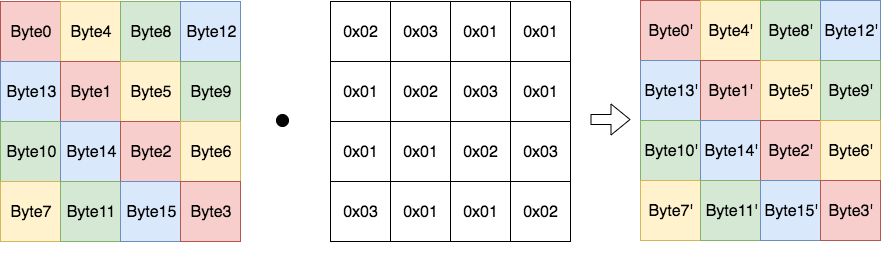
**Figure 2:** Mapping of the byte substitution in the S-Box. The row is chosen by the four most significant bits and the column is chosen by the least significant bits. For example if 0x41 is inputted to the S-Box, the output will be 0x83 (after [6]).

The ShiftRows stage, shown in figure 3, does a simple shuffling of the bytes. By picturing the text being grouped as a 4X4 array of bytes, ShiftRows rotates all bytes right by the value of the row number minus 1. For example, the first row does not shift, the second row shifts right by one, and so on. The bytes shifted out of the array are wrapped around to the left side of the row.



**Figure 3:** Visual diagram showing how the bytes are shifted in the ShiftRows stage. All bytes shifted out of the array to the right are wrapped around to the left of the array.

The output of the ShiftRows stage is fed into the next stage, MixColumns. MixColumns uses matrix multiplication to multiply the resulting 4X4 byte array by an array of predetermined constants.



**Figure 4:** The MixColumn phase computes the dot product of the 16 bytes and an array of constants. The middle block represents the array of constants. These values are standard to the AES algorithm.

The last stage in this phase is the AddRoundKey, which is the same as described in the Initial Round phase. The only difference is the XORing is done with the round key associated with the specific round the algorithm is in.

Final Round is the final phase of the AES encryption algorithm. This phase consists of all the stages in the Round phase except the MixColumns stage. The output is the final ciphertext for the 128 bit block.

AES, although very effective in encrypting data, has some limitations. One of the major ones is the fact that it uses a private key for encryption and decryption. The use of a private key for encryption and decryption means that this private key needs to be securely delivered to each entity involved. Unless the private key can be physically delivered to each entity, the process of secure distribution becomes challenging. This is where the RSA encryption algorithm is used.

*iii. Rivest-Shamir-Adleman*

RSA is an asymmetric key encryption algorithm that uses a private key to encrypt data and a public key to decrypt data, or vise versa [3]. It is built on the foundation of prime number factorization and the challenges behind computing the factorization for large numbers. RSA is often used when information needs to be shared with an entity in which there is no private key shared.

*B. Power Analysis*

Although the algorithms described above have been shown to be secure against cryptanalysis, a new field of attacks has emerged that target the physical hardware of a system instead of the mathematics behind the encryption algorithm. These are referred to as side channel attacks.

Power analysis is a type of side channel attack that focuses on fluctuations in the power consumption of a device. From a power consumption trace, an attacker has the ability to observe specific deviations in power and, since this power is dependent on the data being processed by the device, the attacker can actually decipher bit values from a series of power traces. There are three main types of power analysis, Simple Power Analysis (SPA), Differential Power Analysis (DPA), and Correlation Power Analysis (CPA). Each of these examine variations in power of a device and relate these variations to a system's state and information being processed by the system.

*i. Simple Power Analysis*

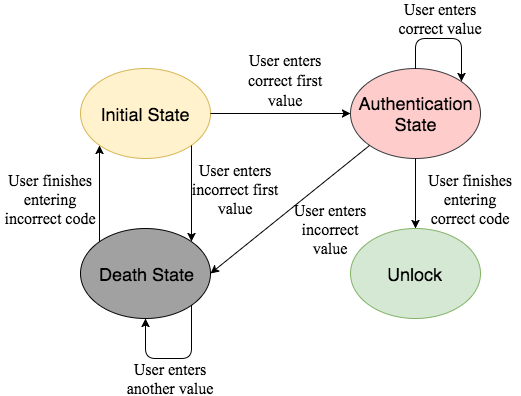
Simple Power Analysis, discussed in reference [3], is the most basic of the power analysis techniques. SPA has two forms, single trace analysis and trace pair analysis. As hinted at by their titles, single trace analysis only uses one power trace and trace pair analysis uses two traces that are compared to one another.

In a device, different applications or tasks require different levels of power consumption. Single trace analysis examines one power trace at a time and picks out these variations in power. With some background knowledge on the device being analyzed, the attacker is able to decipher the state the system is in at specific times. On top of this, if there is minimal noise in the system, bit values can be distinguished using single trace analysis. For example, in some stages of RSA there is a choice of multiplication of values or squaring of values, and this is dependent on whether the data bit is a 1 or a 0. Multiplication uses significantly more power than squaring, therefore when there are spikes in power the attacker can assume the device is in a multiplication phase and can infer that the data bit being processed is a 1. On the contrary, when the power consumption is lower the attacker can assume the device is in a squaring phase and can infer the data bit is a 0.

In trace pair analysis, two power traces are examined together. This is usually done by subtracting one from the other. What is left is a single trace that is relatively flat where the two power traces were similar and sporadic where the traces are different. This data can be used to compare how different inputs affect the power consumption of the device.

SPA is mostly used in addition with other attacks. Many of these side channel attacks are aimed at deciphering a system’s key to be able to decrypt the data being transferred. In a system that has moderate amounts of noise, reading individual bit values from a single power trace is nearly impossible. An example of when SPA is effective is when dealing with an unknown device, an attacker may use an SPA attack to learn what type of encryption algorithm is employed and then use a different attack to attempt to decipher the system keys.

Even though SPA is not used for deciphering keys, it is still a powerful attack. An example of a real-life SPA attack would be an attack on a security pin pad. A pin pad is a type of digital lock that requires the user to enter the correct code for it to be unlocked, and is represented by the state diagram in figure 5. If an attacker conducted an SPA attack on the device, given that each state will have a distinct power consumption, they could monitor how each of their inputted guesses affects the state of the device. The attacker will repeatedly make guesses on the correct code and observe the system's reaction to each guess. A change in power consumption will relate to a change in the state of the device. Through guess and check, and by knowing the power consumption of each state of the device, the attacker can execute multiple rounds of trial and error until the correct code is found.

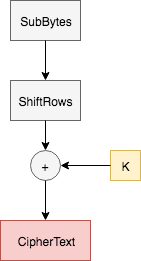


**Figure 5:** State machine of the pin pad being attacked. Because each state will have different power consumptions, these varying power consumptions can be monitored and used to attack the device.

Although SPA is a very effective attack to observe state changes in a system, it is not particularly effective in deciphering bit values. An easy defense to prevent SPA would be to add noise to the system. This would add minor fluctuations in the power trace that would make it challenging to pick out particular bit values.

*ii. Differential Power Analysis*

Differential Power Analysis is a type of power analysis that uses statistical methods to compare a very large set of power traces, pick out correlations between them, and use the correlations to decipher specific bit values. The idea behind DPA is to target a specific round of encryption, this round differs depending on the encryption algorithm used, and makes guesses to what the subkey value is. The following explanation, discussed in [3], will be aimed at an attack on the AES encryption algorithm.



**Figure 6:** Final round of AES encryption algorithm. This is normally where where a DPA attack targets because the ciphertext is known and the only operations on the text are XORing and byte shifting.

A DPA attack usually targets the output of the SubBytes stage of the final round of the encryption algorithm, as shown above in figure 6. The reason the last round is targeted is because the MixColumn stage is not present in the final round of AES.

The main idea behind a DPA attack is to collect power traces, guess the 8-bits of the key value, and then using that, making a prediction on the output of the S-Box. Because the attacker is unable to physically see the actual output of the S-Box to confirm whether the prediction is correct or incorrect, the power traces need to be used to extract this data.

A DPA attack is executed as follows. First, the device is fed arbitrary inputs as plaintext and the plaintext is encrypted using the encryption algorithm. The ciphertext is recorded along with the power consumption trace of the device. Table 2 shows the resulting collection of data. In general, the noisier the system the greater the number of power traces that need to be collected.

|  |  |
| --- | --- |
| Symbol | Description |
| j | Encryption run number |
| CTj | Ciphertext of encryption run, j |
| Powj | Power trace of j |
| K | Key byte guess |
| I | Predicted S-Box output according to K |
| S | Selection function output (Most Significant Bit) |

**Table 1:** Key mapping symbols to their description for the following DPA explanation.

|  |  |  |
| --- | --- | --- |
| j | CTj | Powj |
| 0  1  2  3  .  .  .  n | 0010 0001  1000 1110  1011 0000  0010 1111  .  .  .  1010 0011 | Pow0  Pow1  Pow2  Pow3  .  .  .  Pown |

**Table 2:** For j encryption runs, power traces, Powj, are recorded along with their corresponding ciphertext, CTj.

Once these power traces are collected, the attack moves into the key byte guess phase. Here, the attack uses the key byte guess, K, and the ciphertext, CTj, to predict what the S-Box output, I, should be. Next, a selection function, S, is chosen. In this case the selection function takes in the predicted S-Box output and selects the most significant bit of the data. The power traces associated with the S-Box output guesses are used to check if the prediction was correct. All the power traces that, according to the key guess, should have a selection function output of 1 are placed in a set, Set 1. The same thing is done for the power traces that should have a selection function output of 0.

The power traces in each set are then averaged together. If the sets are split correctly, there is a specific point at which all power traces in Set 1 are computing a 1 value and all power traces in Set 0 are computing a 0 value. These two traces are then subtracted from one another. The variation in Set 1 and Set 0 will result in a spike after the subtraction. This spike indicates a correct key guess.

|  |  |  |  |
| --- | --- | --- | --- |
| j | K | I | S |
| 0  1  2  3  .  .  .  n | 0000 0000 | 0101 0111  1110 0011  1001 0101  0010 0000  .  .  .  1110 0101 | 0  1  1  0  .  .  .  1 |
| 0  1  2  3  .  .  .  n | 0000 0001 | 1000 0011  1010 1011  0000 1101  0111 0001  .  .  .  1100 1010 | 1  1  0  0  .  .  .  1 |
| .  .  . | | | |
| 0  1  2  3  .  .  .  n | 1111 1111 | 0111 0001  0011 1101  1001 1001  1010 1011  .  .  .  0001 0000 | 0  0  1  1  .  .  .  0 |

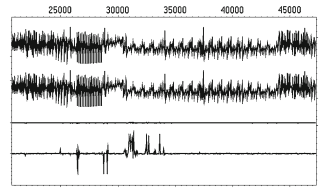
**Table 3:** Process of taking trace j and using key guess K to predict the output of the S-Box I. Selection function, S, in this case is simply selecting the most significant bit of I.

|  |  |  |
| --- | --- | --- |
| K | Set 0 | Set 1 |
| 0000 0000 | Pow0, Pow3, … | Pow1, Pow2, … , Pown |
| 0000 0001 | Pow2, Pow3, … | Pow0, Pow1, … , Pown |
| .  .  . | | |
| 1111 1111 | Pow0, Pow1, … , Pown | Pow2, Pow3, … |

**Table 4:** The power traces are separated into two sets. One for the traces that, according to the key guess K, should have a 0 as their selection function output and one for traces that should have a 1. The power traces in each set are averaged together and then the set averages are subtracted from each other to examine correlation.

If the guess is incorrect, by the laws of probability, there will be close to 50 percent incorrect guesses in each of the sets. When the power traces in the sets are averaged, they will have generally identical results. Then when they are subtracted from each other, there will be no resulting spike. This will result in a flat line throughout the entire power trace representing an incorrect key guess.

Since the attacker is focusing on one byte of the key at a time, they will run through all the power traces collected 28 times for each key byte guess until the correct byte guess reveals itself. Once this is completed for one byte of the key, the attacker targets a different S-Box output and repeats the process until the entire last round subkey is uncovered. From there the attacker can work backwards through the key expansion algorithm to recreate the master key of the system. Once this master key is uncovered, the system is compromised.



**Figure 7:** The top two power traces represent the average power consumption of Set 0 and Set 1. The third power trace shows the result of the subtraction of the top two traces. The fourth trace is a zoomed in version of the third. The spikes relate to differences in the average power traces of the sets and indicate a correct key guess (after [3]).

*iii. Correlation Power Analysis*

Another type of power analysis technique is Correlation Power Analysis (CPA). CPA is a version of Differential Power Analysis that focuses on all the values in a byte rather than just working with the output of a selection function [3][7]. The attack makes guess on what a subkey byte would be, uses the known text to model what the power consumption would be if that key byte guess was correct, and then compares this model to the actual power trace. The subkey byte guess with the highest correlation is going to be the most likely key.

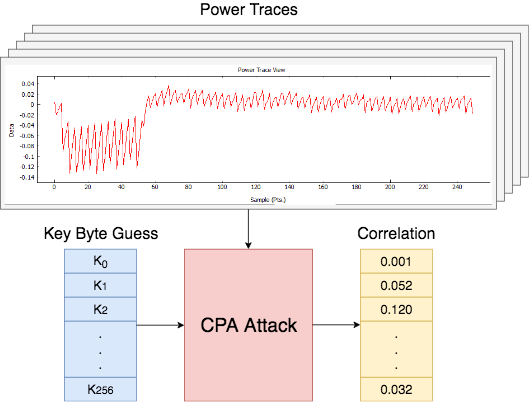


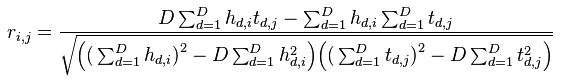
Figure ??: CPA attack takes in the power traces, recorded from the device while it runs encryption, and key byte guesses. The attack outputs correlations for each guess and the guesses with the highest correlation are the most likely key bytes.

This attack is made up of 4 different aspects. The first is creating a model in which to compare recorded power consumptions with generated power consumptions, from using the key byte guess and the known text. In this case the most effective model is the Hamming Weight model. Hamming Weight is the number of 1’s existing in a bit stream. For example:

A larger number of 1’s being processed is going to correlate to a higher power consumption at that time. The Hamming Weight allows a comparison of data being processed and the power being consumed because of that data.

The next aspect of the CPA attack is to record the power consumption traces of a device while it encrypts a number of random data streams. The power traces recorded will be compared against the modeled power consumption of the key guesses. The noiser the system the more traces that will need to be recorded.

Once all the traces are recorded the next step is to use the traces to conduct the attack. This is done by examining one byte of the key and taking guesses on what that key is. Since the attack only looks at a byte at a time, there are only 2^8=128 possible values of that byte. The attack iterates through each of these guesses and uses the known text to generate a model of what the power consumption would be if that key byte guess is correct. This is done for every possible value of that byte. These models are then compared to the actual power trace using the equation below.



* t → power trace
* D → total power traces
* d → individual trace
* j → sample point in trace
* i → key byte guess
* h → power estimate
  + h(d,i) power estimate for trace d and guess i

Using this equation the attack finds the highest value of *| r(i,j)* |. From there, it finds the highest *| r(i)* | and the key byte guess that gives you that value is going to be the most likely key.

The final step of the attack is to repeat this process for each byte of the key. At the end, the collection of key byte guesses is the most likely round key and from there the attacker can work backwards to find the overall encryption key. Once this key is obtained the system is then compromised.

*C. Defence Implementations*

These power analysis attacks have the ability to compromise a system without performing any cryptanalysis on the encryption algorithm used. Systems that run these encryption algorithms need to examine not only the encryption algorithm used but also the implementation of the algorithm on the device. The following subsections discuss different implementations of security measures to defend against these side channel power analysis techniques.

*i. Tamper Resistant Systems*

One way to defend against an attack on a system is by being able to sense when an attack is happening. The design proposed in [8] uses a Security Primitive Controller (SPC) to implement the systems security operations. Their design also includes a System Security Controller (SSC) whose main purpose is to detect when the system is being attacked. This controller uses sensors connected to the battery, the bus, other communication channels, and other security primitives to detect out of the ordinary behaviors resulting from fault injections or abnormal operations.

The SSC has the ability to interrupt the SPC. Depending on the irregularity the SSC senses in the system, it will instruct the SPC to perform a specific function. This type of system defends against power analysis by detecting when the power consumption of the device is being monitored by an outside source. The SSC will detect this intrusion and will instruct the SPC to halt the encryption module. By halting the encryption module, the power traces collected by the outside entity will not be sufficient for data analysis.

Being able to alert a system when it is being attacked is a very viable defense against attacks, but it comes at a cost. Implementing this extra SSC controller and the sensors that go along with it will increase the hardware and complexity of the system. Along with this, having the SSC constantly polling these sensors to detect irregularities will use extensive amounts of power.

*ii. Randomness*

All defenses implemented to protect a system from an outside observer are aimed at masking the processes being executed in the system. Along with this, all side channel attacks can be executed because of consistent patterns in data. One way to combat these attacks is by adding an element of randomness to the timing of the system. An element of randomness adds unpredictability that cannot be modeled by an attacker. This approach is discussed in [3].

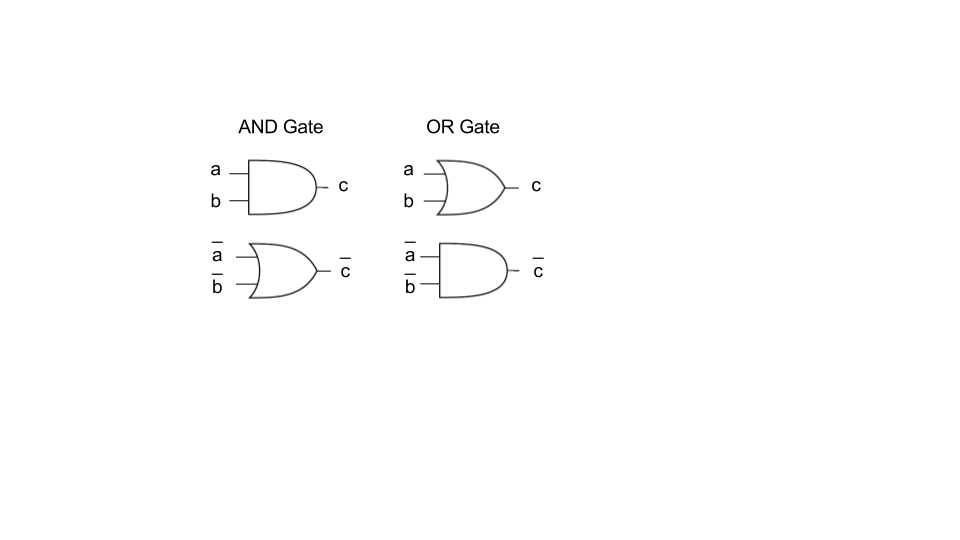
Attackers need to observe a system for a period of time before being able to successfully attack it. They look for patterns based on time intervals and correlate these patterns to specific functions in the system. By randomizing the amount of clock cycles before an instruction executes, there will be no clear time interval for each operation. Irrelevant instructions can be inserted into the instruction path to delay the execution of the thread. An attacker will not be able to tell which instructions are placeholders and which are relevant to the thread’s execution. This defence works especially well against power analysis. The varying number of clock cycles needed for execution throws off the alignment of power traces making them significantly more challenging to compare to one another.

Randomizing clock cycles is an easy design to implement in a system and is very low cost in terms of hardware. The downside is that it hurts the performance of the processor. When the processor is executing all these pointless instructions, it is not processing the meaningful ones. These relevant instructions get delayed to a later time and the system does not run as efficiently.

*iii. Dynamic and Differential Logic*

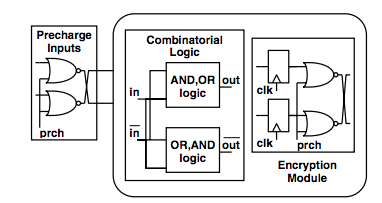
Another defence against a power analysis attack is Dynamic and Differential Logic (DDL). DDL allows the evaluation of each gate to have exactly one transition. This occurrence of a single bit flip is independent of the data being processed by the gate. By only having one bit flip in each gate, regardless of the data being evaluated, that cell will always consume approximately the same amount of power. To do this, DDL uses two main ideas. One, for each gate being evaluated, also evaluate its complement. The true gate and its complement together make a DDL cell. The second main idea is, the outputs of each gate are precharged to a set value before evaluation takes place. The precharge phase is used to initialize every gate output to a constant initial value. Because each gate output is set to the same value, when a DDL cell is evaluated, either the true gate or its complement will have a bit flip, but never both.

Sensor Amplifier Based Logic (SABL) is a type of Dynamic and Differential Logic and is used on Application Specific Integrated Circuits (ASIC) [1]. In SABL, there are two phases, a precharge phase and an evaluation phase. The output of each individual gate is ANDed with the precharge value. Reference [1] explains, “whenever the precharge signal is 1, the outputs are predischarged to 0 independently of the input-values.” The problem with this approach is the design area is essentially quadrupling. For every gate in the design, one, its complement gate needs to be added and two, an AND gate has to be added for every true gate and for every complement gate.



**Figure 8:** Representation of an AND gate and an OR gate in WDDL format. Both the true value and its complement are evaluated.

Another approach to DDL is Wave Dynamic Differential Logic (WDDL). WDDL is an implementation of DDL in which the precharge phase is initiated by forcing all inputs to 0. By doing this, each gate will evaluate to 0 and will have a precharge value of 0 at their outputs. This 0 will propagate through the entire design, like a wave [9]. WDDL uses the same idea of having a complement gate for each true gate, making a WDDL cell, and therefore, will still only have one bit flip for every evaluation of that cell.



**Figure 9:** The precharge signal is applied to the NOR gates along with the inputs to the encryption module. If the precharge signal is 1, the input to the encryption module will always be 0. This triggers the wave which propagates throughout the system (after [1]).

The advantages of using WDDL are one, it can be implemented on an FPGA, and two, it does not require the addition of an AND gate for every gate in the design. The precharge phase is instead initiated by adding *n* NOR gates, where *n* is the number of inputs to the design. Each NOR gate takes in an input to the design and the precharge signal. Figure 9 shows a representation of this set up. Whenever the precharge signal goes high, the output of the NOR gate will always be 0 and this will be propagated into the design. This forces all gate outputs in the design to 0 [1]. By precharging before evaluation, each cell will only have one transition of 0 to 1 during the evaluation phase and will in turn have a constant power consumption that is independent of the data.

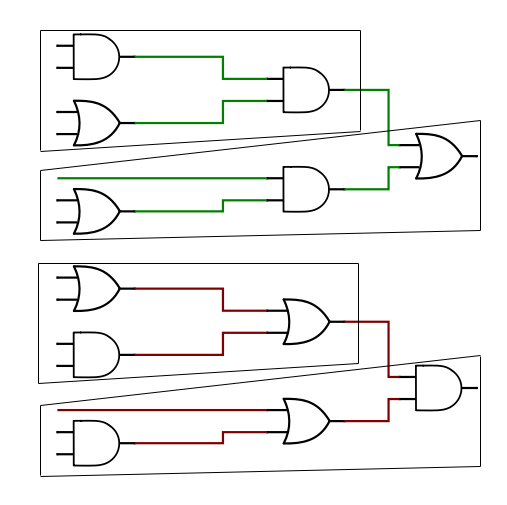
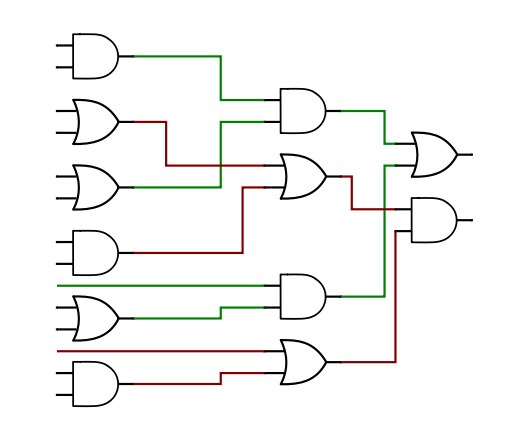
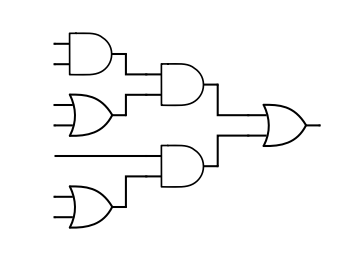
*D. FPGA WDDL Implementation*

A WDDL design can be implemented on an FPGA as described in [9]. There are a couple challenges when implementing such a design on an FPGA that make it more difficult than an implementation on an ASIC. One of the challenges is, when programming all these gates on the FPGA, the FPGA will try to optimize the design. It will look to combine these gates into the same lookup tables (LUT) to reduce the area of the design. This is an issue when trying to implement a WDDL design because in WDDL the main goal is to balance load capacitances of each cell along with the routing between cells. Optimizing the design by combining gates into the same LUT will result in unbalanced load capacitances, which will in turn result in fluctuation in power consumptions related to the data being processed.

In WDDL there are two aspects of load capacitance that need to be considered. The first is the load capacitance of the LUT and the second is the load capacitance of the routing from LUT to LUT. Fortunately, two LUTs with the same number of inputs and outputs have identical load capacitances. The aspect that the designer needs to worry about is the load capacitance of the wires routing the output of LUTs to different LUTs. An approach for balancing this capacitance between a gate and its complement gate is by instantiating each gate to its own individual LUT and then combining these two LUTs into a single slice in the FPGA to represent a WDDL cell. A slice is a section of an FPGA consisting of a number of LUTs, multiplexers, and registers.

This is an effective approach because as stated before, the LUTs will have identical load capacitances. Along with this, one of the properties of a slice is that the routing from one LUT to the output of the slice is the same distance as the routing of the other LUT to the output of the slice. Since they are the same routing distance they will have the same load capacitance within the cell. A constant load capacitance within the WDDL cell will result in a constant power consumption by the cell.

Another challenge of a WDDL design on an FPGA is the amount of area it takes up. If a designer were to implement a WDDL design such as the one described above, one, the instantiation of each gate to an individual LUT would greatly increase the area the design takes up, and two, by also including each gates complement, the area would double on top of that. This becomes a serious concern when the FPGA is suppose to handle other applications along with encryption.



**Figure 10:** By implementing WDDL on the initial circuit shown in the first figure, result shown in the second figure, multiple gates can be grouped together to potentially be instantiated to the same LUT. This grouping reduces the area of the design.

One way to work towards reducing this amount of area in a WDDL design is by combining multiple gates into a single LUT. The figure above shows a small section of a design with three OR gates and three AND gates, along with its WDDL implementation. As seen in the figure, the gates generating the true values can be separated from the ones generating their compliments. By doing this, a designer can implement these two different components in four LUTs with four inputs and one output each. Now, instead of having twelve LUTs in the design, there are only four.